2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

# OBJECT-ORIENTED IMPLEMENTATION OF FIELD ARTILLERY TACTICAL DATA SYSTEM

by

Mustafa Eser

September 1993

Thesis Advisor: C. Thomas Wu

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY<br>2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release;<br>distribution is unlimited |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Computer Science Dept.<br>Naval Postgraduate School<br>6b. OFFICE SYMBOL (if applicable) CS | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Monterey, CA 93943-5000 | 7b. ADDRESS (City, State, and ZIP Code)<br>Monterey, CA 93943-5000 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>8b. OFFICE SYMBOL (if applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State, and ZIP Code) | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO. |

11. TITLE *(Include Security Classification)*
OBJECT-ORIENTED IMPLEMENTATION OF FIELD ARTILLERY TACTICAL DATA SYSTEM (U)

12. PERSONAL AUTHOR(S)
Eser, Mustafa

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT *(Year, Month, Day)* | 15. PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM 09/92 TO 09/93 | 1993, August 26 | 162 |

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

| 17. COSATI CODES | | | 18. SUBJECT TERMS *(Continue on rever if necessary and identify by block number)* |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Field Artillery Tactical Data System, Object-Orieted Programming, Object-Oriented Database Management System |

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
The U.S. Army lacks a single automated fire support system. The goal of Army's ongoing project of Advanced Field Artillery Tactical Data System (AFATDS) is to integrate all of its fire power under a single automated system to provide an efficient fire support in the battlefield. AFATDS is being implemented using the language ADA for battalion and above level. The problem for this research is to implement AFATDS for battalion (just for technical fire direction) level and below. In addition, we want to add a Graphical User Interface (GUI), use modern software engineering principles and add multitasking.

The approach taken was to apply object-oriented paradigm for the design and development of the battery level of AFATDS using Microsoft Windows' operating environment which provides (non-preemptive) multitasking and a GUI, and Borland C++ as the development tool.

The results are as follows: The battery level software of AFATDS is implemented. The GUI provided a better interface which facilitates easier training [Ref. 17]. Multitasking allowed multiple firing missions to execute concurrently which was not possible with BCS. Object-oriented features of Borland C++ provided 60% improvement for GUI development than traditional programming languages.

| | | |
|---|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED | |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>C. Thomas Wu | 22b. TELEPHONE *(Include Area Code)*<br>(408) 646-3391 | 22c. OFFICE SYMBOL<br>CS/Wq |

DD FORM 1473, 84 MAR
83 APR edition may be used until exhausted
All other editions are obsolete
SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

# *Object-Oriented Implementation of Field Artillery Tactical Data System*

by
*Mustafa Eser*
*First Lieutenant, Turkish Army*
*B.S., Turkish Land War Acedemy, 1987*

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
September 1993

Author: [illegible]
*Mustafa Eser*

Approved By: [illegible]
*Dr. C. Thomas Wu*, Thesis Advisor

[illegible]
*Dr. David K. Hsiao*, Second Reader

[illegible]
Prof. Ted Lewis, Chairman,
Department of Computer Science

# ABSTRACT

The U.S. Army lacks a single automated fire support system. The goal of Army's ongoing project of Advanced Field Artillery Tactical Data System (AFATDS) is to integrate all of its fire power under a single automated system to provide an efficient fire support in the battlefield. AFATDS is being implemented using the language ADA for battalion and above level. The problem for this research is to implement AFATDS for battalion (just for technical fire direction) level and below. In addition, we want to add a Graphical User Interface (GUI), use modern software engineering principles and add multitasking.

The approach taken was to apply object-oriented paradigm for the design and development of the battery level of AFATDS using Microsoft Windows' operating environment which provides (non-preemptive) multitasking and a GUI, and Borland C++ as the development tool.

The results are as follows: The battery level software of AFATDS is implemented. The GUI provided a better interface which facilitates easier training [Ref. 17]. Multitasking allowed multiple firing missions to execute concurrently which was not possible with BCS. Object-oriented features of Borland C++ provided 60% improvement for GUI development than traditional programming languages.

# TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

## TABLE OF ABBREVIATIONS

| | |
|---|---|
| AFATDS | Advanced Field Artillery Tactical Data System |
| ACCS | Army Command and Control System |
| API | Application Interface |
| BCS | Battery Computer System |
| BLOBs | Binary Large Objects |
| DBMS | Database Management System |
| DLL | Dynamic Link Library |
| DPMI | DOS Protected-Mode Interface |
| FA | Field Artillery |
| FATDS | Field Artillery Tactical Data System |
| FDC | Fire Direction Center |
| FDO | Fire Direction Officer |
| FFE | Fire for Effect |
| FS | Fire Support |
| GDI | Graphics Device Interface |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| LOC | Line of Code |
| MBPDC | Model-Based Project Development Cycle |
| MTO | Message To Observer |
| O-O | Object-Oriented |
| OOA | Object-Oriented Analysis |
| OOD | Object-Oriented Design |
| OODA | Object-Oriented Domain Analysis |
| OODBMS | Object-Oriented Database Management System |
| OOL | Object-Oriented Language |

| | |
|---|---|
| OOP | Object-Oriented Programming |
| OORA | Object-Oriented Requirement Analysis |
| ROF | Record of Fire |
| TOT | Time -on-Target |

## ACKNOWLEDMENTS

I would like to thank Dr. Wu for introducing me to the Object-Oriented paradigm, 1LT Cince (Turkish Army) for contributions with his discussions and experiences on my program & interface design, 1LT Ozen (Turkish Army) for providing me technical documents and CPT Freeman (US Army) for his precious help for providing materials of fire direction.

Particularly I would like to thank my fiancèe, Zeliha, for her inspirational support which helped me to finish this thesis on-time.

# I. INTRODUCTION

## A. BACKGROUND

The mission of artillery is to provide and coordinate devastating fires, giving the maneuver commander the overwhelming combat power to win decisively and quickly anywhere and any time. Modern artillery has improved its effectiveness using new technologies with weapon and $C^3I$ systems.

Technology developments, productivity, and battlefield concepts always force $C^3I$ components to work in an integrated environment to attain victory in the warfare. The current leading field artillery tactical automated system essentially based on TACFIREs/ LTACFIREs which are battalion level systems. The sub-systems of a TACFIRE/ LTACFIRE are Battery Computer Systems (BCSs) and various interfaces. It is an isolated system from other automated systems and services. Next step is to integrate all the tactical data systems of the army, namely Army Command and Control System (ACCS). Field Artillery Tactical Data System will be the subsystem of ACCS.

Advanced Field Artillery Tactical Data System (AFATDS) which is an ongoing project being a sub system of ACCS is aimed to realize artillery portion of the integration. Since AFATDS had not been fielded yet, the concept of FATDS is preferred for the same scope of AFATDS for this thesis research. This integration will provide:

- Centralized database which aims dumping databases to other computers to provide backup, exchanging data between relevant nodes and reducing data redundancy.
- Faster fire support.
- Saving resources.

Field Artillery Tactical Data System (FATDS) is composed of several layers and independent interfaces. Layers are:

- Battery level
- Battalion level

- Corps level

Interfaces are:

- Digital message devices
- Screens

No matter how modern armies prepare themselves for future conflicts, there will be always surprises and unconditional requirements. Reprogrammability of automated systems provides flexibility to new requirements of a battlefield. At this point software part of automated systems needs programming paradigms which provide high degree of software engineering goals which are: [Ref.21: p.65]

- Modifiability: *The software system should be easily modifiable without causing other errors or complexity.*
- Efficiency: *The software system should operate using the set of available resources, which are time and space, in an optimal manner.*
- Reliability: *The software system should operate for long periods of time without human intervention.*
- Understandability: *It is essential for modification and project management because of the need for coordination.*

In addition to those goals software developers have been experimenting with various methodologies to achieve some of the following goals: [Ref.18: p.16]

- Precise specification of the design.
- Shorter code development phase.
- Easier development of code.
- Reusability of code.

### 1. Advanced Field Artillery Tactical Data System

AFATDS is an ongoing project for total fire support system for U.S. Army, with its fielding in FY'95. AFATDS will align targets with fire units and munitions and help manage survey and meteorological operations, movement and positioning and process of managing logistics. AFATDS will take into account all available fire support means, to include attack helicopters, tactical air, naval gunfire and offensive electronic warfare. It will employ sophisticated routines to recommend the right systems for the targets under analysis and facilitate the synergistic integration of efforts. The heart of AFATDS' development is software.

> The heart of AFATDS' development has been the software. The first step in the AFATDS program was to establish the command and control functions of the fire support system, a detailed analysis that identified the inputs, the processes and the outputs. The results of this analysis have driven the AFATDS software. [Ref.23: p.39]

Major advantages of AFATDS in terms of software and hardware are seen as: [Ref.23: p.39-41]

- Easy upgrades
- No more Mutual Support Units (MSUs)
- Comprehensive Functionality
- Interoperability
- No AFATDS military occupational speciality (MOS) necessary
- Common hardware Army-wide
- Large screen display
- No more dedicated shelters
- Fire Support Elements (FSEs) linked from battalion through corps
- Lab-Top computers for Fire Support Officers (FSOs) and commanders

A key aspect of AFATDS software will be its interoperability. As a component of ACCS, AFATDS will interface with maneuver, intelligence, air defence and combat service support systems. It is also being designed to interface with the Air Force and Marine automated systems and with the German (Adler) and British (BATES) systems. These interfaces will significantly enhance the total integration of the joint and combined team. [Ref.23: p.40]

AFATDS has a distributed database architecture. The database of each AFATDS node will be maintained continually with the latest information.

To maintain a full integration, portability and easy maintenance, AFATDS is using DoD's standard programming language, ADA. AFATDS is conceptually evaluated in 1989 and scheduled to be completed in FY'93 with fielding to the total force scheduled for FY'95.

### 2. Battery Computer System

BCS is defined as follows:

An automated data processing system located in the firing battery. It consist of three major components: the battery computer unit, the power distribution unit, and 1 to 12 gun display units. It is used to compute accurate firing data and as digital communications interface. [Ref.10: Glossary-2]

### 3. Motivation for Object-Oriented Approach

My motivation for O-O approach in general:

- Object-oriented approaches encourages the use of *modern* software engineering technology.
- Object-oriented approaches promote and facilitate software reusability.
- Object-oriented approaches facilitate interoperability.
- When done well, object-oriented approaches produce solution which closely resemble the original problem.
- When done well, object-oriented approaches result in software which is easily modified, extended, and maintained.
- Object-orientation improves traceability.
- Object-orientation improves the conceptual integrity of both the process and the product.

## B. OBJECTIVES

The objective of this thesis is to create the battery level software of Field Artillery Tactical Data System analyzing the benefits of object-oriented implementation of both programming and database of the project in an environment which provides multitasking and graphical user interface.

## C. ORGANIZATION

Chapters are organized into two groups: Programming language and database. Chapter II evaluates Object-Oriented Programming development languages and tools briefly.

Chapter III addresses the design features of programming phase. Chapter IV is a comparison of the Object-Oriented language implementation of subject vs. conventional programming languages. Chapter V evaluates the possible database management options for FATDS. Chapter VI addresses the design of the selected database management option. Conclusions and future works on FATDS are summarized in Chapter VII.

# II. THE PROGRAMMING ENVIRONMENT, LANGUAGE AND APPLICAITON DEVELOPMENT TOOLS FOR FATDS

The concept of Object-Orientation may not give any idea about its performance over other paradigms without considering programming environment, language and application development tools.[1] For this research I have chosen PC-based Microsoft Windows 3.1 as the programming environment, C++ as the Object-Oriented Language (OOL), and Borland C++ & Application Framework as the application development tools. Since the purpose of this research is to focus on the programming performance benefits of object-oriented implementation, the best selection of those elements is not discussed.

Object-Oriented Design (OOD) and Object-Oriented Analysis (OOA) of FATDS will be discussed on the next chapter. Following sections discuss the characteristic of the selected elements.

## A. PROGRAMMING ENVIRONMENT

When it comes to portability issue of a software, there are questions to be answered about programming environment. What will the application be used for? What kind of terminals will be used? Who will be the users? Which operating system will the application work on? What kind of devices will the application work with? Will the application work stand-alone or in networked environment? Does the application have to work with heterogenous subsystems in an network? There are many questions that a programmer faces with. I have taken PCs as the main hardware unit. Table 1 displays the system requirements of the O-O FATDS. The reasons for selections this environment are:

- In addition to technologic developments, economical and administrative obligations makes all the uniform systems heterogenous. Especially huge systems do not tend to keep themselves uniform. PCs are unique examples to represent the variety of users, hardwares and software. Though its heterogeneous structure creates some problems - like training, hardware, software maintenance, either single or in a networked media

1. When an object-oriented system is under construction, the architecture is foremost in importance, just as it is in all other engineering disciplines. The development process is adapted to the architecture, and the tools are adapted to the process. [Ref.15: p.26]

PCs are forcing companies to pay more attention to them. PCs word is developing with its new features and object-orientation is claimed for being familiar with these complexity.

- PCs are more available computers to ordinary people. This means more people are familiar with PCs (which is important from the user's train point of view) and there are sophisticated application tools for PCs too.
- Its software and hardware are more economical than work stations.
- This selection satisfies the objective of this research.

TABLE 1: SYSTEM REQUIREMENTS OF O-O FATDS

| | Application Development$_{min}$ | Application development$_{optimal}$ |
|---|---|---|
| CPU | IBM-386 | IBM-486 |
| Display | VGA, 2 color | SVGA, 256 color |
| Sound | - - | - - |
| Operation System | MS-DOS 5.0 & MS-Windows 3.1 | MS-DOS 5.0 & MS-Windows 3.1 |
| Memory | 2 MB RAM | 8 MB RAM |
| Hard Disk Space | 80 MB | 120 MB |

## 1. Hardware Basics

Hardware capabilities may boost the performance of the application, but there is no direct relation with the programming style for this research. Board, CPU, caching, display and other I/O devices have direct relations with either operating system or application requirements.

## 2. Windows Basics

The general structure of Windows programming naturally fits the OOP style. Everything on the screen is defined as a window, because each object has to have some common features. Each window inherits some of its features from other windows and add some specific ones. The data goes with a window is encapsulated into the window object.

Window objects are polymorphic, they can all receive common messages and take action that is appropriate for that window. [Ref.25: p.4]

#### *a. The GUI*

Does a user, who is supposed to use a computer, have to learn what the computer is? Or, should the computer be designed regarding a specified user group? Practically both questions have positive answers. Needs and technology determine each of its ratio in an application.

Most computer users are familiar with Windows' GUI and all Windows applications have consistent interface. Consistency is one of the important factors which helps a user to adopt a new application in shorter time.

Though every environment provides different GUI in different degrees of graphicism Windows provides all the basic GUI tools. These are the most common graphical elements:

- Icons
- Fonts and text formatting
- Charts and graphics
- Menus

One of the graphical elements of Windows Application Interface (API) is controls which represents an event or information source for events. Controls are mainly used for user input. The leading feature of controls is to make the user remember some information but not to memorize for recalling. The controls are:

- Buttons
- Check boxes
- Combo boxes
- Static/dynamic edit fields
- Radio buttons
- Custom controls

There are high level graphical elements too:

- Drag-and-drop mechanism
- Hot spots in text or graphics to do something when selected.

Those were the just the elements of Windows GUI. The problem is to produce something that the user will understand and use it easily.

#### b. Event-Driven Applications

The only communication link between a user and a terminal is Input / Output (I/O) actions. Procedural interfaces provide I/O in either command line or question-and-answer form. In either case, the application gets some input, goes off to process it, and produces output. The application is the boss, and the user acts as an intelligent database server. Menu-driven applications takes some database burden of the user. [Ref.17: p.39]

Event-driven applications act on events coming from a variety of sources. The events may come from controls that the user can activate, from other programs, or from devices such as the clock or communications port.

Event-driven applications are not specific only to Windows, but almost all Windows applications are based on this style. Event-driven programming brings a different approach to the design and implementation of an application. Events leads the way, so establishing a GUI generally is the first phase of implementation. Just a GUI, without a real code behind it, serves as a prototype of the application. This eliminates other third party tools for rapid code prototyping.

#### c. Multitasking

Windows 3.1 provides a non-preemptive multitasking ehvironment. This feature is one of the key factor for the end-user. The user can run more than one application simultaneously.

#### d. Device Independence

This feature is important for both user and programmer. The device dependency is hidden away in the Windows' Graphics Device Interface (GDI) and deice drivers that are either bundled with the retail Windows or supplied by the device manufacturer. Windows program developers can ignore about devices, they know that their application will work with all the devices that are introduced to Windows.

> For programmers in non-Windows environment, program output can be the most challenging task of the entire application. Idiosyncrasies of a wide variety of devices must be discovered, resolved, and properly handled in the code. For small developers, just obtaining the necessary manuals and use of devices for testing can prohibitively expensive Under Windows, all these problems simply goes away.[Ref.19: p.78]

### *e. Elements of a Windows Application*

A typical Windows applications has these elements: A main window contains a title bar, menu and a client area. The client area serves as a canvas that the application can draw on. Main window may have child windows. Child window looks sometimes as a control or sometimes another client region. Dialog boxes are for either information display or input purposes. Dialog boxes contain a number of controls, such as buttons, radio boxes, etc.

### *f. Dynamic Link Libraries (DLL)*

DLLs are one of the advanced features of Windows. Normally most functions have been statically linked to applications. When a library or object code is used the executable versions of the functions contained in the code are brought in to executable file. [Ref.20: p.48] Since this is a static linking, multiple versions of the same function may exist in the executable file. Any modification on the library functions require relinking of the application. So the whole application is replaced.

Dynamic linking occurs at run-time. Application is just introduced to the function and in which. DLL file its implementation is placed. The executable file does not carry the object code. When the executing program makes a call to a dynamic link function, Windows checks to see whether the function is already in memory. If it is in memory Windows increases the in-use counter for this library by one and passes execution to the new memory location. If the function is not already in memory, Windows tries to find a file, with a .DLL extension, that contains the desired function. Windows then loads the function into memory. [Ref.20: p.49]

The only disadvantage of DLLs is that .DLL files should be ported with the application while the advantages of DLLs are:

- Downsize the executable code size and compile-link time. E.g., OFIRE.EXE file was more than 400K lon before using .DLLs, and it become almost 150K long executable file.
- Provide the application to run on computers that have a limited amount of memory.
- Ease the maintenance of the code. When a DLL file is updated, it is enough to replace that .DLL file without linking the whole code.
- Make interchangeable modules.[Ref.25: p.66]

## B. PROGRAMMING LANGUAGE

I have selected C++ as an OOPL. It is not the only OOPL, but it is one of the OOPLs that supports all the O-O features. One of the leading reasons which makes me select C++ is its commercially availability.

> For better or worse, the real winner today is C++, which attracts large groups of programmers at the largest companies and is used on thousands of projects in industry. [Ref.15: p.26]

Stroustrup lists the language features of C++ as:

- C-based, object-oriented extension of C.
- Class concept and class-based object-oriented paradigm.
- Data abstraction and data encapsulation facilities.
- Mechanism for data abstraction hierarchies.
- Strongly typed.
- Function and operator overloading.
- User-controlled memory management.
- Facilities modeling multiple inheritance.
- Support polymorphism.
- Type-safe linkages.
- Abstract classes.
- Exception handling mechanism.
- Used for general purpose applications as well as simulations development.

- Can be use to model conceptual solutions.

## C. APPLICATION DEVELOPMENT TOOLS

Borland provides all the tools that is necessary to produce Windows programs.

### 1. Application Development tools

Borland C++ 3.1 and Application Framework has the following tools:

- An ANSI C and C++ global optimizing compiler
- A DOS protected-mode interface (DPMI) compiler and programmer's platform
- A Windows-base integrated development environment (IDE)
- A graphical source browser (ObjectBrowser)
- A utility for tracking Windows messages (WinSight)
- A debugger for DOS and Windows applications (Turbo debugger)
- A profiler for DOS and Windows (Turbo Profiler)
- The Turbo Assembler
- The Resource Workshop
- A library of C++ classes to simplify Windows application development (ObjectWindows)
- The EasyWin library for porting DOS programs to Windows
- The Turbo Vission application framework for DOS applications
- Source code to the runtime library

### 2. Language

Borland compiler runs in protected mode, so large programs and libraries do not require extensive swapping. Precompiled header are supported. Because the headers for Windows programs tend to be quite large (windows.h alone is over 120K), this Borland C++ feature can increase compile times by a factor of ten.[Ref.19: p.xviii][Ref.25: p.5]

Borland IDE editor provides to open multifiles simultaneously as well as large files.

# III. OBJECT-ORIENTED PROGRAMMING DESIGN OF FATDS

## A. INTRODUCTION

Object-oriented programming should follow Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD). Though O-O paradigm goes back two decades, OOA and OOD methodologies are not mature enough. [Ref.14: p.22] Discussions are still continuing about a perfect methodology for both OOA and OOD. The goal of OOA is still the same: the development of an accurate and complete representation of the problem domain. [Ref.14: p.26] The problem is that there is no standard OOA methodology. I have selected Coad & Yourdon OOA [Ref.6] and Booch OOD [Ref.4] to build basis for O-O methodologies of this research.

### 1. Coad and Yourdon OOA

This model defines the problem domain in five consecutive steps where each step builds on the previous step. The steps are:[Ref.14: p.27]

- Define objects and classes: *Look for structures, other systems, devices, events, roles, operational procedures, sites, and organizational units.*
- Define structures: *Look for relationship between classes and represent them as either general-to-specific structures.*
- Define subject areas: *Examine top level objectives within whole-to-part hierarchies and mark these as candidate subject areas. Refine subject areas to minimize interdependencies between subjects.*
- Define attributes: *Identify the atomic characteristics of objects as attributes of the object. Also look for associative relationships between objects and determine the cardinality of those relationships.*
- Define services: *For each class and object, identify all the services it performs, either on its own behalf or for the benefit of other classes and objects.*

Coad and Yourdon uses the following tools for OOA:

- Class and object diagram
- Object state diagram
- Service chart

## 2. Booch OOD

Booch is the pioneer of OOD. Booch's methodology was originally ADA language specific, but he adopted his methodology to O-O completely. He recommends not a specific ordering of analysis plus design phases. The strategy should be "analyze a little, design a little" [Ref.4: p.201]. Because there is no certain border between analysis and design of a problem to jump from one to another. Design process should be worked with iteratively and incrementally, augmenting formal diagrams with informal techniques appropriate to the problem at hand. Booch's four major design steps are [Ref.14: p.32]:

- Identify classes and objects: *Identify key abstractions in the problem space and label them as candidate classes and objects.*
- Identify the semantics of classes and objects: *Establish the meaning of the classes and objects identified in the previous step using a variety of techniques, including creating "scripts" that define the life cycles of each object from creation to destruction.*
- Identify relationships between classes and objects: *Establish class and object interactions, such as patterns of inheritance among classes and pattern of cooperation among classes and patterns of cooperation among objects. This step also captures visibility decisions among classes and objects.*
- Implement classes and objects: *Construct detailed internal views of classes and objects, including definitions of their of their various behaviors (services). Also, allocate objects and classes to modules.*

Booch uses the following tools for OOD:

- Class diagrams and class templates
- Object diagrams and timing diagrams
- state-transitions diagrams
- operation templates
- module diagrams and templates
- process diagrams and templates

## 3. My OOA and OOD Strategy

As it is indicated above I carried out analysis and design together. The reasons are:

- There is no distinct border between analysis and design. Switching to design to late may cause waste of resources while switching too early may cause not to identify the problem. It is more flexible to do them together.
- The design may not reflect the capabilities of the system environment or the requirements of the user completely. Instead of using external software prototyping

tools, GUI-based prototyping which could be performed at various steps of system development, provides realistic and early evolution of the design. Each time after receiving feed-backs from the consumer, it could be necessary to modify the analysis again.

- There are diverse design options for the project. Not all the design options are derived from the analysis. At a point in the project it would be necessary to modify the design and the analysis process as well.
- There may appear some constraints and force to change to alternate designs. E.g., size of code may force the system hardware's memory limitations.

Above conditions are valid not only for O-O methodologies, but also for non-object-oriented ones. After evolution of a system, if a modification is required, then it will be necessary:

- To add a/some new part(s) into the system and/or,
- To throw away a/some part(s) of the system and/or,
- To modify a/some part(s) of the system

Object-oriented systems adds another option for creation or modification of a system: *Reuse*. At the beginning of a large project development or on small projects the role of reusing may not be so important, but during the development of phase and for large scaled projects reusing begins to save resources. Assuming that old components of the system are well tested, other objects which are created by inheriting those old objects are most likely be trusted objects. It will take shorter time to test new objects.

First product after a brief analysis was the GUI of the project. The design of GUI is an application-independent technology for which a number of useful libraries of reusable software components have been developed. So, GUI's development was faster than the applications. FIGURE 1 on page 16 show the cycles for O-O FATDS project development. This cycle is named as *Model-Based Project Development Cycle* (MBPDC). The product(s) is always tested with the real-life requirements during the development cycle, even though the product(s) is not a complete one during the process. First analysis helps to form classes and objects to create the first form of the product which is just the GUI. Modeling and testing begins with this GUI which serves as a prototype. This is more than a usual prototype which simulates a real-life application, but it is the real-life application. Modeling leads the design, so the analysis.

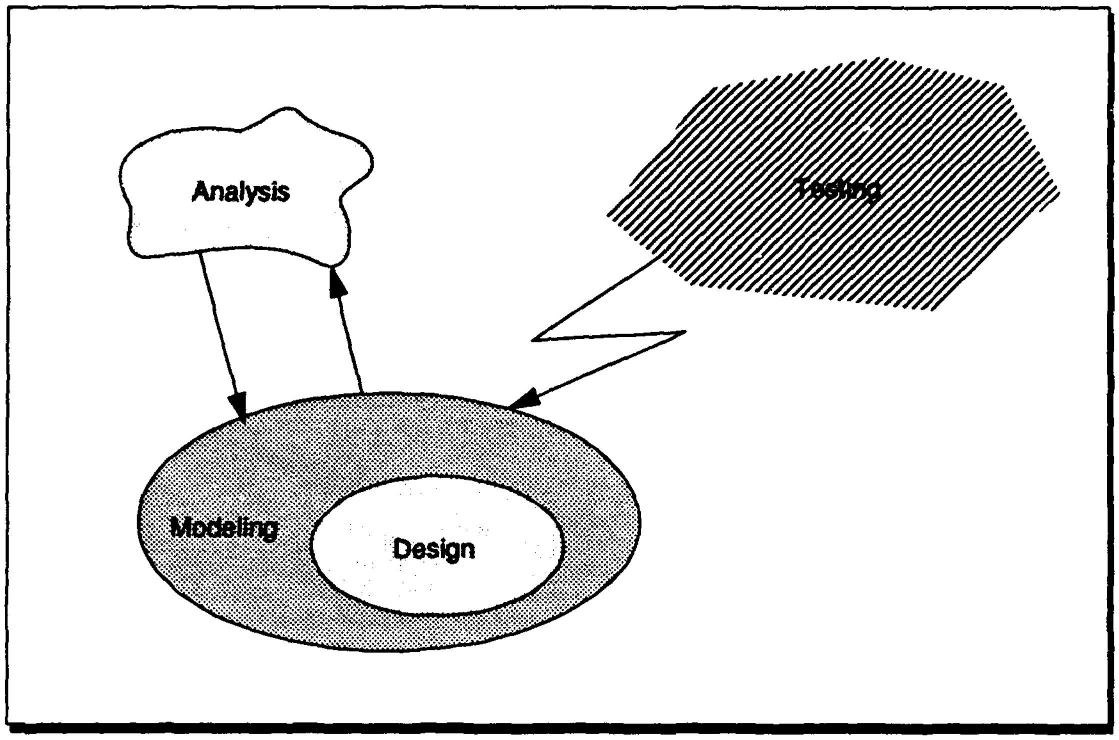


**Figure 1: Model-based project development cycle for O-O FATDS**

There is always a working model. Functionality of the model is limited during the development cycle and it completely matures at the end of the MBPDC..

The nature of event-driven programming and its compatibility with OOP helps to establish separate or independent event modules. This event inter-independency and reusing makes the modification easier at any time of MBPDC.

## 4. Naming Conventions

For a more readable code I have used long meaningful names for variables, enumeration types, classes, and objects etc. E.g., SubFireCallWndCls and RadioGrid.

I have also embedded small identifiers into the names to distinguish them from others. In addition, I named an instance of a type or class almost the same with its instantiator, e.g., SubFireCallWndCls is a class type and SubFireCallWnd is instance of that class. Table 2 on page 17 displays a list of embedded codes which are used for naming in this research.

TABLE 2: EMBEDDED CODES FOR NAMING

| | |
|---|---|
| `*Cls` | Class defination |
| `*DlgCls` | Dialog window class defination |
| `*Dlg` | Dialog object |
| `*WndCls` | Window class defination |
| `*Wnd` | Window object |
| `Button*` | Button control object |
| `Check*` | Check box control object |
| `Combo*` | Combo box control object |
| `Edit*` | Edit control object |
| `List*` | List box control object |
| `Radio*` | Radio box control object |
| `h*` | Any kind of object handler |
| `ID_*` | Identifier for window objects |
| `*Type` | Enumeration type definition |

## B. ANALYSIS

### 1. FATDS System Requirements

FIGURE 2 on page 18 displays a block diagram for Firing Support (FS) portion of FATDS. The basic function of the system is to make the guns fire with computed data against the targets on the battlefield which are asked from artillery to be destroyed.

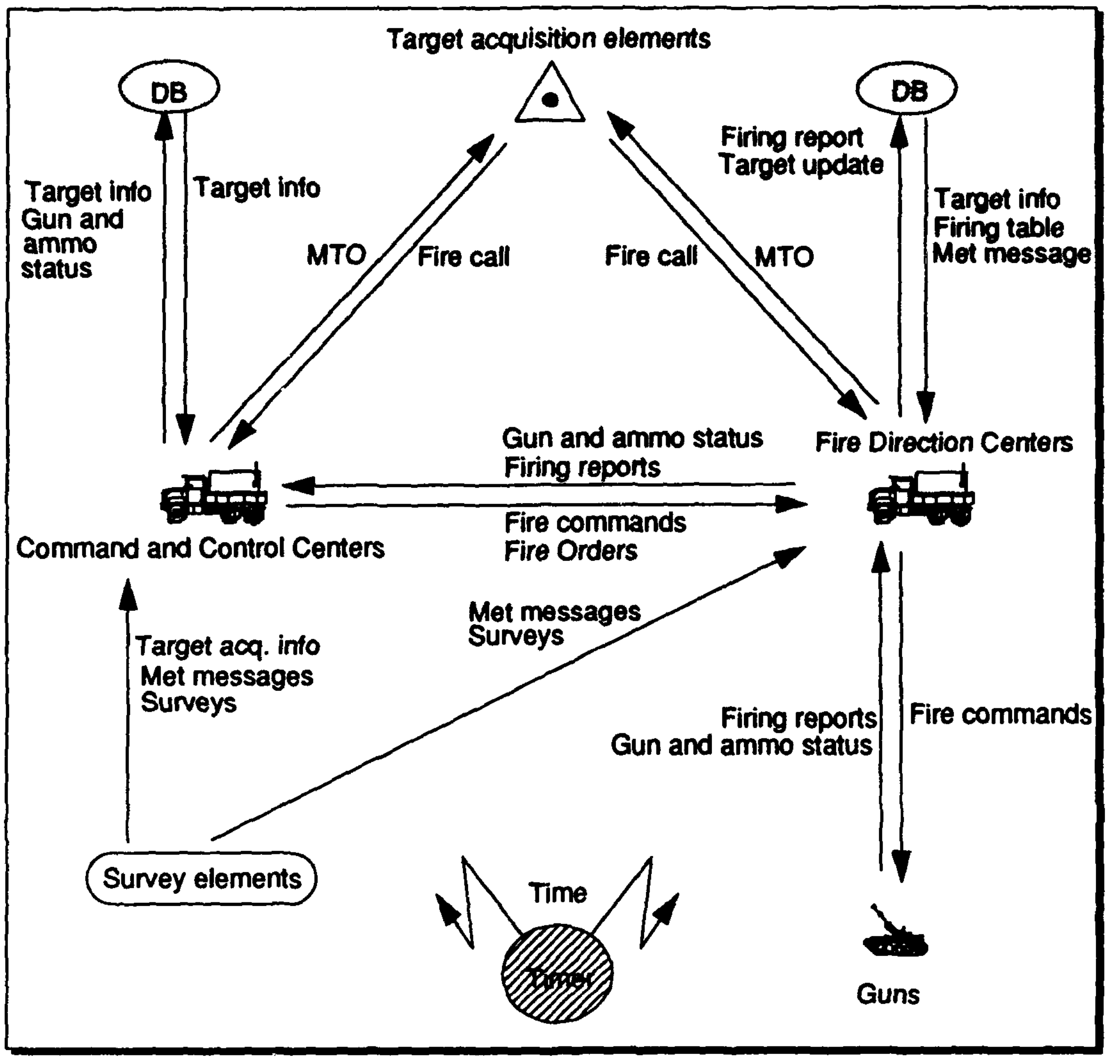


**Figure 2:** Block diagram for Field Artillery Tactical Data System / FS

Each of the block elements could represent more than one of that element and in diverse organizational structures. The automation of the each structure may differ too. Regarding the communication tools of any kind and battlefield irregularity, the system should keep to work with acceptable functionality. The major system elements are:

- Guns: *They are organized in platoons, batteries or battalions as basic firing units. The number of guns and models of them vary.*
- Fire direction centers: *The element of a command post consisting of gunnery and communication personnel and equipment by means of which the commander exercises*

*fire direction and/or fire control. The fire direction center receives target intelligence and requests for fire and translates them into appropriate fire direction [Ref.11: p.G-8]. FDCs are functional at platoon, battery and/or battalion levels.*

- Target acquisition elements: *There are various target acquisition elements. FA battalion S2, targeting officer, fire support team, aerial fire support observers, combat observation/lasing team, survey platoon, weapons-location radar, moving-target-locating radar and electronic warfare are all the elements for target acquisition.[Ref.7: p5-1]*
- Command and control elements: *From Fire Direction Officer (FDO) at platoon / battery level to higher tactical headquarter who plans and controls the fire are included in this block. Their role is tactical rather than technical. These elements make decisions about "Who will fire?", "Which target(s) will be fired?", "What kind of weapon(s) & ammunition(s) combination will be formed?", "When will be the firing time? Different levels of command may have different control responsibilities on various operations.*
- Survey elements: *These are technical elements too. Survey elements provides met messages, target intelligence, and land survey support for the FDC and command control elements.*
- Database: *Local or centrally located, distributed or single database system is the main information source of the system. To avoid waste of resources, central databases are encouraged.*
- Timer: *This is just a control element which sets the timed fires or other events. All the elements should follow a central time function for synchronization.*

The following inputs exists:

- Fire call
- Fire order
- Fire commands
- Fire orders
- Firing reports
- Firing table
- Gun and ammo status
- Message to observer (MTO)
- Met messages
- Subsequent fire calls
- Status reports
- Subsequent fire call
- Surveys
- Target acquisition
- Target info
- Target lists
- Target update

• Time

### 2. Defining the Boundaries of the Problem

Field Artillery Tactical Data System is sub-part of the army's integrated battlefield automation program. Since this is a huge and sophisticated program this research has captured only Field Artillery (FA) Battery Fire Direction level regarding its FS portion. FIGURE 3 on page 20 displays the block diagram of battery level tactical data system / FS.

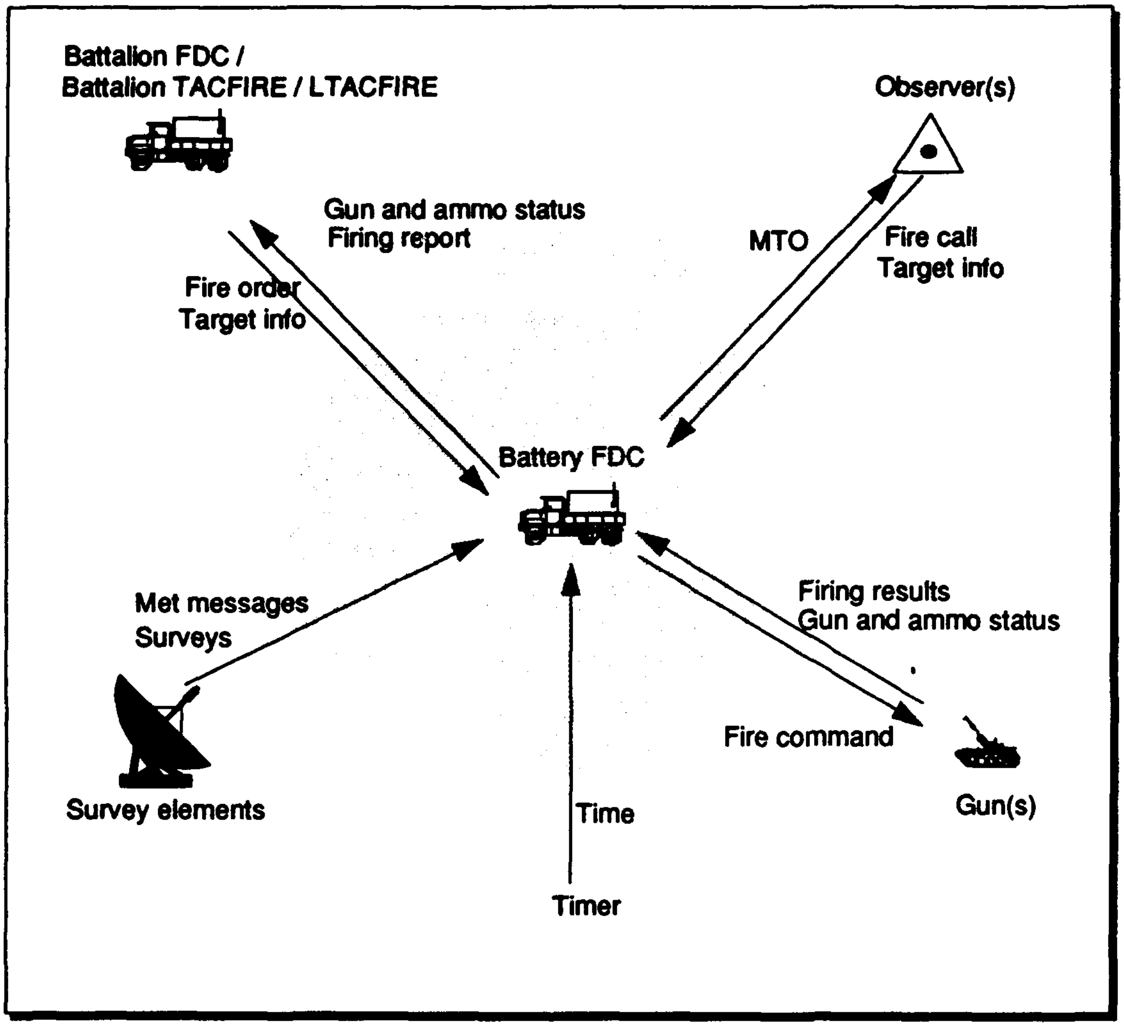


**Figure 3:** The block diagram for FATDS / FS at Battery level

FIGURE 4 on page 21 displays the block diagram for battery fire direction system. There are two major functional units, *firing chart* and *computation*. Firing chart

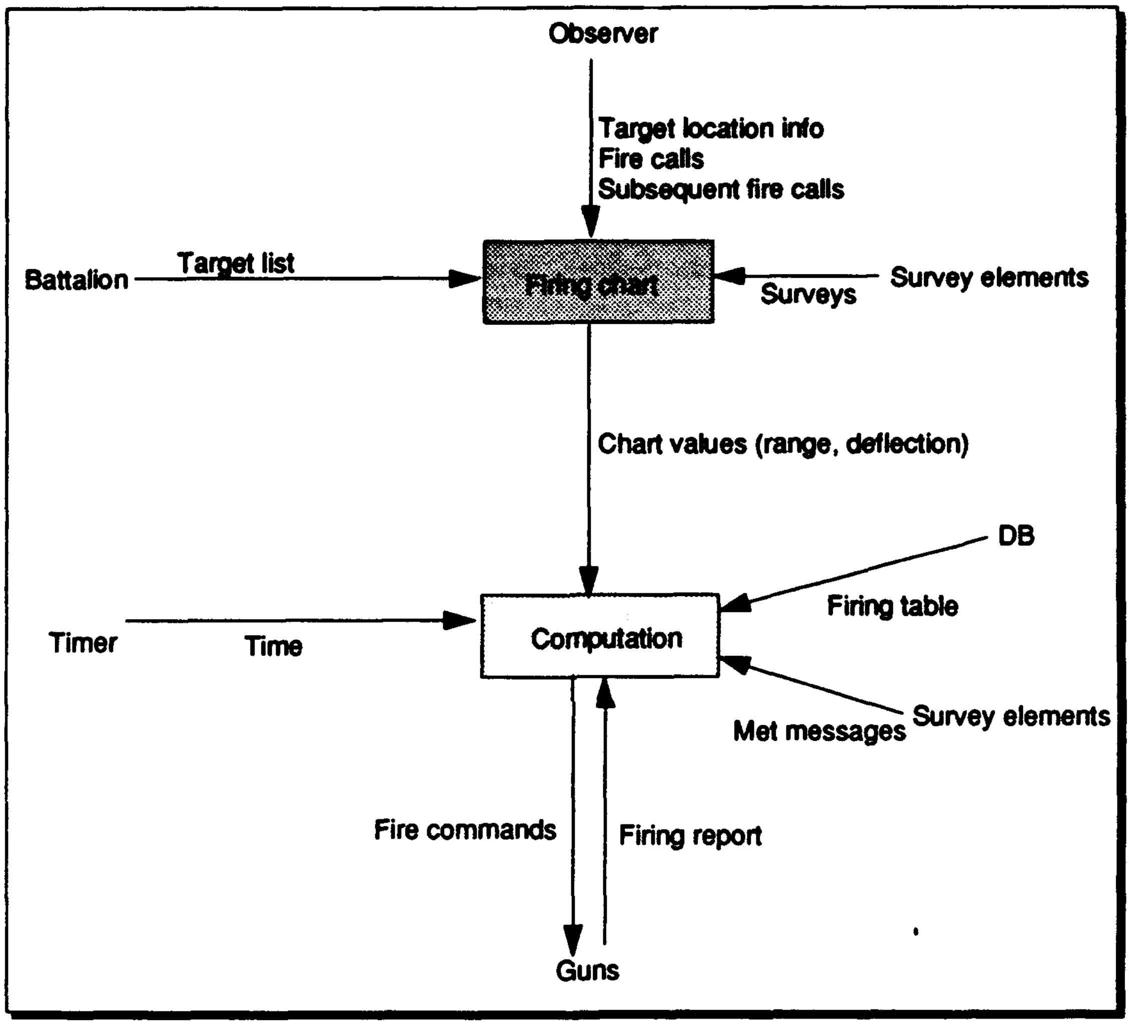


**Figure 4:** The block diagram for battery fire direction system

holds the coordinates of:

- Batteries (gun positions)
- Radars
- Observation posts
- Registration points
- Targets

- Check points

Firing chart produces linear and angular distances of any two points, which are on the chart, for computation.

After having chart values Computation transfers them into firing commands and sends them to the firing elements (guns). Computation reflects its computational results on Record of Fire (ROF) form. Computation produces the firing commands regarding:

- Time: *At my command, When ready, Time on Target (TOT).*
- Firing techniques: *Surveyed firing, copperhead munition firing, nuclear munition firing, smoke projectile firing, illuminating projectile firing, MET+VE technique, registration, Fire for Effect (FFE), special corrections.*
- Firing tactics: *Attack of large targets, target analysis, safety procedures, munition effects.*
- Ballistics (interior and exterior): *Nature of propellant and projectile movement, muzzle velocity, meteorological conditions, dispersion.*

### 3. Defining the Objects and Classes

Followings are the classes derived from the block diagram of the system and/or description of the diagram elements.

- Dispersion
- Chronometer
- Fire commands
- Fire Calls
- Subsequent fire calls
- Fire order
- Message to observer
- Computation
- Firing report
- Gun and ammo status
- Target database
- Firing report database
- Met messages database
- Guns and munitions database

### 4. Define Subject Areas

Classes are grouped into two main subject areas: *Firing chart* and *firing computation*. Those groups will be the basis for the design of GUI. The classes, which have been formed so far, are nothing to do with GUI but they will be re-organized after the events of the GUI is determined.

## C. GRAPHICAL USER INTERFACE DESIGN

So far two functional unit appeared. They will be named as *OFire* for firing computations and *OChart* for firing chart manipulations. GUI design was the backbone of the analysis phase. They have been accomplished together. Since the GUI served as a prototype of the final product, it was useful determine the events for objects, classes and the relationships of them.

### 1. Design Considerations

The design of a GUI is being effected not only by the capabilities of the system's environment, but also by the following criteria.

#### *a. The User*

Since both OChart and OFire are specific applications for a Field Artillery, the number of the users is limited with the FA Fire Direction Center (FDC) personnel. Training requirements are still one of the major issues with the user. Windows GUI environment eliminates some of the training problems and add these features:

- The user does not have to memorize everything. He/she inputs most of the data by selecting, not by entering. He/she is not overloaded with decision making points at the same time, but he/she is asked to do only when necessary.
- The GUI is forgiving for the user decisions, it is possible to take decision steps back. So, the system is less error-prone.
- Even though the user knows nothing about Dos/Windows or any other OS which operates a window style environment, this will help the user to adapt the software in a shorter time than the command line style interface. There is no certain educational background except reading&writing.

### *b. Consistency*

One of the most important issues with GUI is consistency. Each part of the applications should be consistent with:

- Other issues of the user's way of life: *Button shapes and behaviors, sounds and graphics for warnings, etc., should be consistent with the user's daily life.*
- Other applications in that OS environment: *The user should transfer some of her/his experiences from one application to other. This helps to shorthen the training and adaptation period, besides prevent some errors which are source from controversies.*

### *c. Screen-Layout Issues*

Menu system, naming terminology, semantic grouping of items, fonts and use of color are main points of screen-layouts. All of these factors contribute to success of a software, so the victory.

### *d. Customizing*

To allow a user to customize an application makes easier to use that application in accordance with that user's condition only. To provide standardization and easier adaptation of users to the applications customization is not allowed for OChart and OFire applications.

### *e. Input and Output Devices*

Main input device for Windows applications is mouse as well as keyboard. Voice activated commands may be used with some limitations too. Lessening the role of keyboard for data entry decreases the training needs and increases the performance of the application

### *f. On-Line help and Tutorials*

Application should feed the user with necessary information which is about what he/she is doing, or to do etc., at a time when he/she needs. Windows allows a standard style help system which is context-sensitive and topic-navigated.

#### g. *Error Handling*

People are tend to make errors even if they are trained on that topic. Windows event-driven environment makes it easier to add check points and warnings for each reaction of the user.

### 2. OFire

All the windows and dialog boxes designed for the OFire application are displayed on APPENDIX-D.

### 3. OChart

All the windows and dialog boxes designed for the OChart application are displayed on APPENDIX-D.

## D. OBJECT-ORIENTED DESIGN (OOD)

### 1. Identify Classes and Objects and Their Semantics

The classes have been formed with their actual names and grouped into programming modules. Not all the classes have been formed at the same time and those classes are the last phase of the development.

#### a. *The Classes for OFire*

(1) bursts.h

- SheafWndCls
- BurstsWndCls

(2) button2.h

- Button2Cls

(3) command.h

- CommandWndCls
- ChronoWndCls
- FireCommandsWndCls

- InfoWndCls
- SubFireCallWndCls

(4) fire.h

- FireCallDlgCls
- PolarWndCls
- ShiftWndCls
- GridWndCls
- SuppressionWndCls
- FireOrderDlgCls
- MTODlgCls
- SubFireCallDlgCls
- TOTAtTimeDlgCls
- TOTAfterTimeDlgCls

(5) globals.h

- ComputationCls

(6) report.h

- ReportWndCls
- TextWndCls

(7) select.h

- SelectionWndCls

#### b. *The Classes for OChart*

(1) abslist.h

- genAbstractList: Implements an abstract class of linked lists.

(2) Chart.h

- GridSetupDlgCls
- TargetSetupDlgCls
- GunSetupDlgCls
- NewEntryDlgCls
- NewFromEntryDlgCls
- GunInputDlgCls
- TargetInputDlgCls
- TargetDBDlgCls
- GunDBDlgCls

(3) gen2list.h

- genOdList
- genUdList

(4) globals.h

- GlobalCls

(5) gundb.h

- GunDBCls

(6) info.h

- SelectionToolCls
- DistanceToolCls
- NewToolCls
- NewTToolCls
- SearchToolCls
- PickToolCls
- InfoWndCls
- ToolBarWndCls
- CanvasWndCls

(7) Targetdb.h

- TargetDBCls

### 2. Identify Relationship Between Classes and Objects

FIGURE 5 on page 28 and FIGURE 6 on page 29 display the class relationships. In those figures system and user-defined classes are distinguishable. All the dialog or window classes which helps to develop GUI, inherit most of their behaviors from their parent classes.

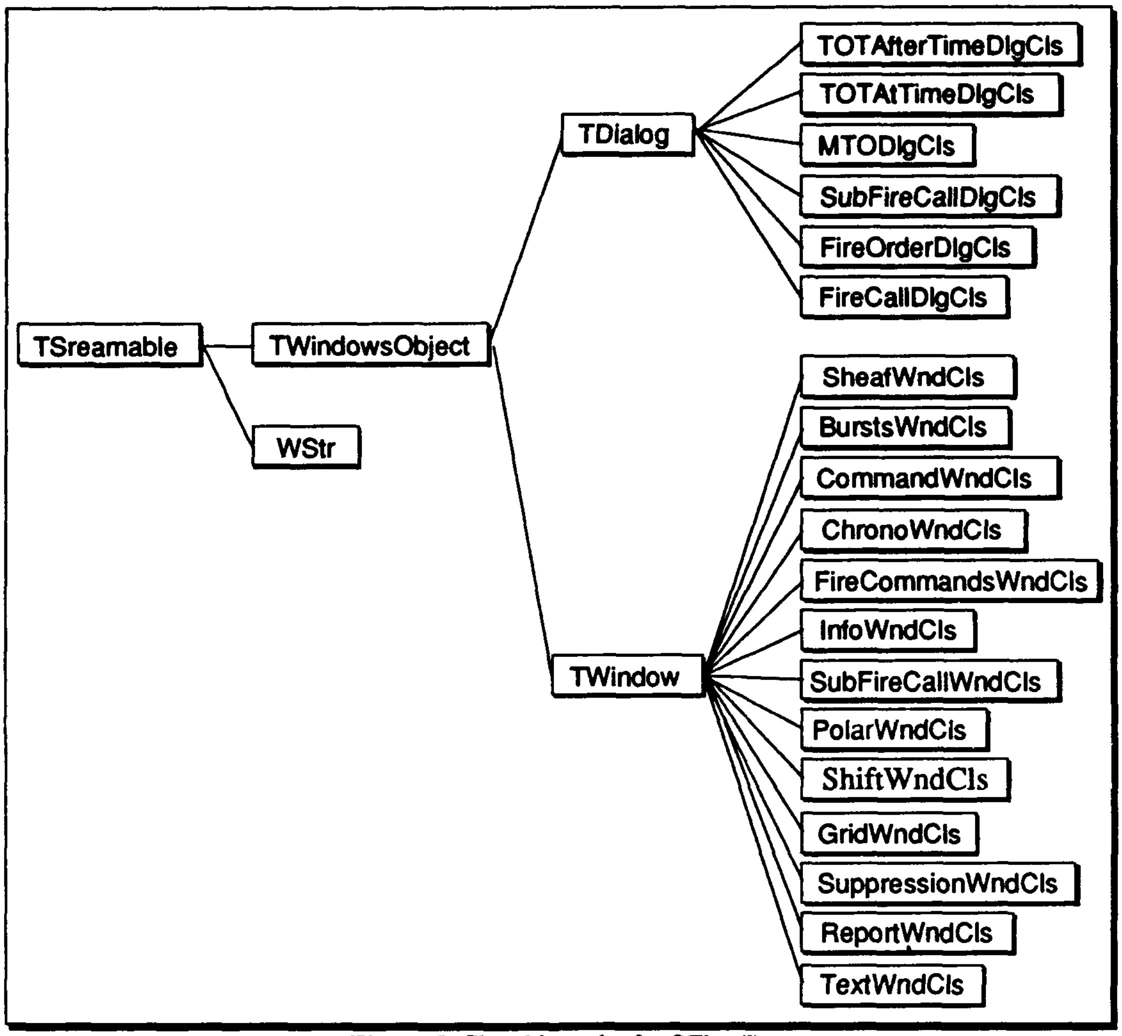


**Figure 5:** Class hierarchy for OFire (i)

### 3. Implement Classes and Objects

Complete listings of header files are provided on APPENDIX A and APPENDIX B.

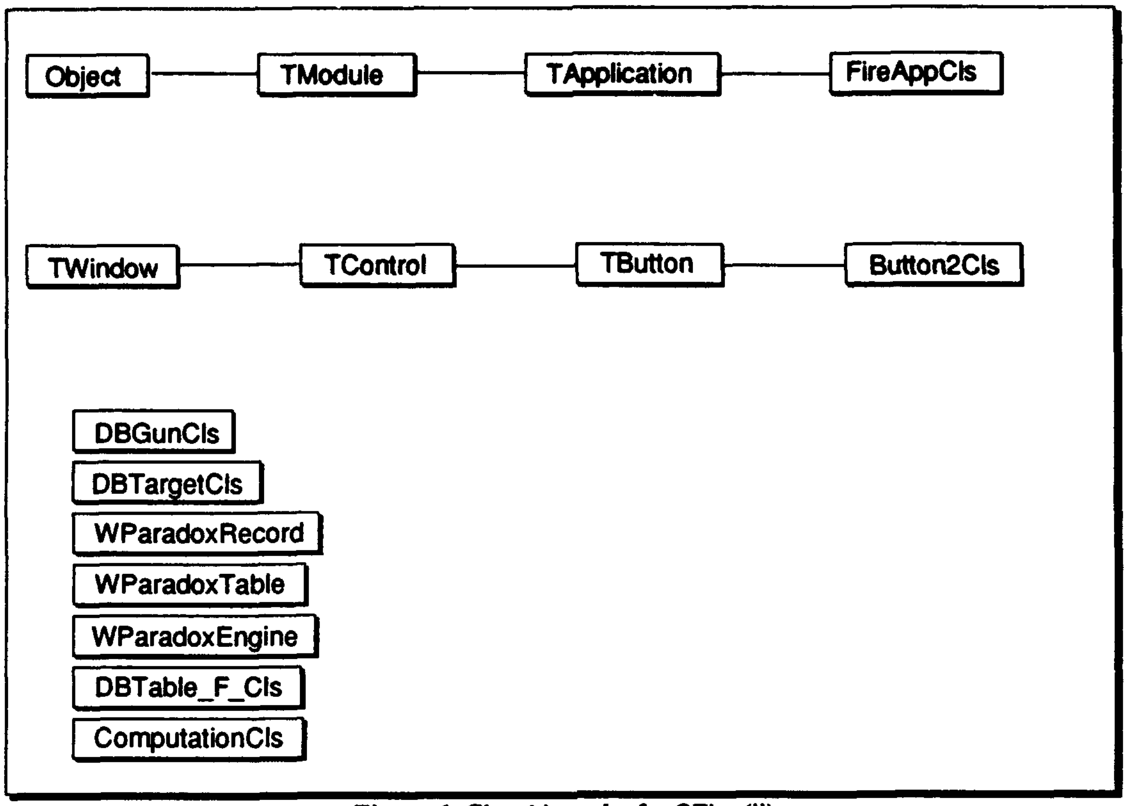


**Figure 6:** Class hierarchy for OFire (ii)

## E. FILE ORGANIZATION

### 1. File Organization for C++ Windows Programming

FIGURE 7 on page 30 shows a sample file organization for a a simple C++ Windows program. Every class is defined in a header file and its implementation is placed in source files. One of the exception to this rule is Main file. There are two class definitions for a main Windows program and a main program function. One of them inherits from TApplication class and it registers the instance of the application to Windows system. The other one inherits from TWindow or TDialog class and it creates a window for the application. The declarations of those classes are generally placed in Main Source File since they will not be called or instantiated by anyone. Only main program functions instantiates them and use it.f

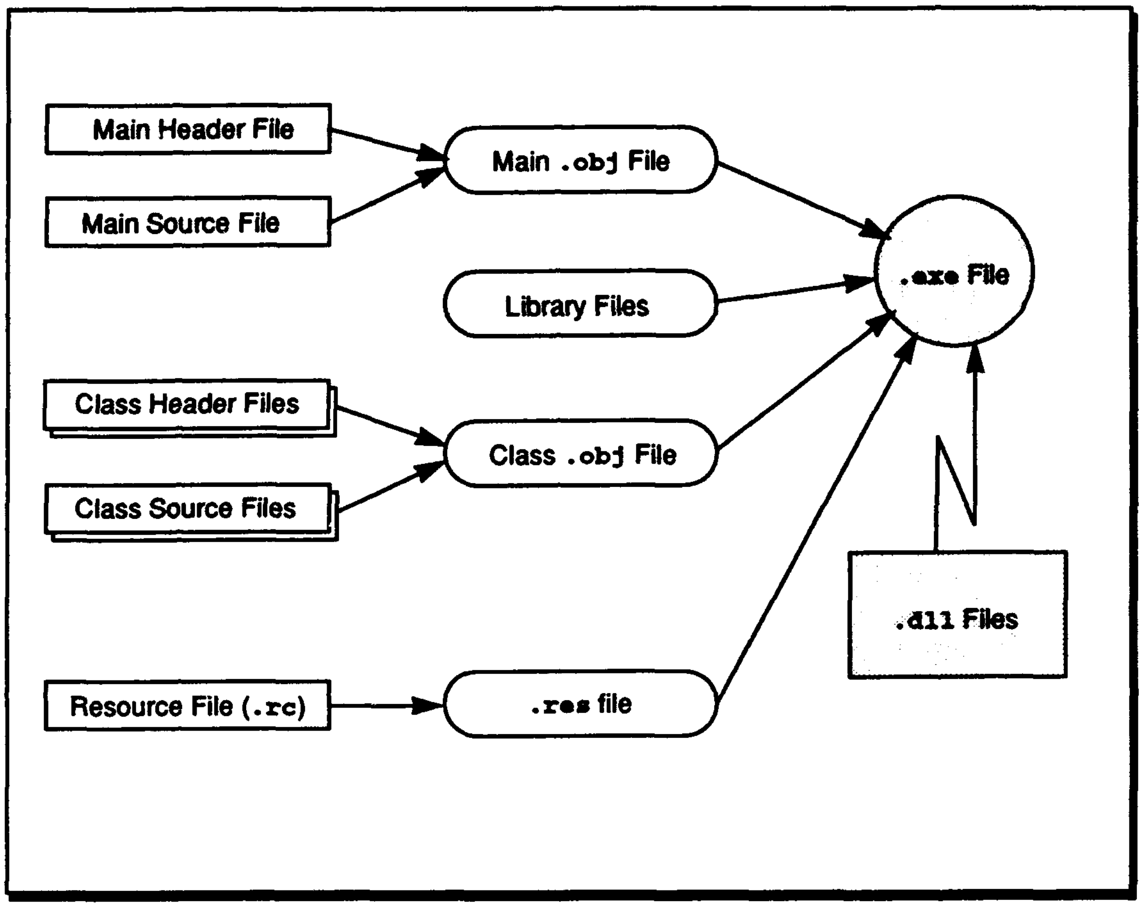


**Figure 7:** C++ development file organization for Windows environment

Main Header File is used for some definitions, and dialog box class declarations. If some objects are placed in .DLL files, its header files are still kept like other built-in class libraries. The only difference for their usage is that, it should be told to the compiler the implementation of any class or function is placed in .DLL files. That specific.DLL file will only be searched during run-time.

Another file which is specific to Windows is resource file. It is a text file and holds the resource descriptions of:

- Accelerators
- Bitmaps
- Cursors
- Dialog boxes
- Fonts
- Icons
- Menus
- String tables

Res⌐ urce file can be created either manually or with Resource Workshop. It is not in a specific language, though it is in a special text format. Borland C++ compiler compiles it with a special compiler and links it to the code. All the resource code is placed in the executable file in separate modules. This allows the Resource Workshop application to open and modify the resources of an application without disturbing the executable's functionality. Eliminating compile and linking efforts, this makes the maintenance easier from the resource point of view.

## 2. FATDS's File Organization

The file organization of FATDS is shown on FIGURE 8 on page 32 and FIGURE 9 on page 33 .

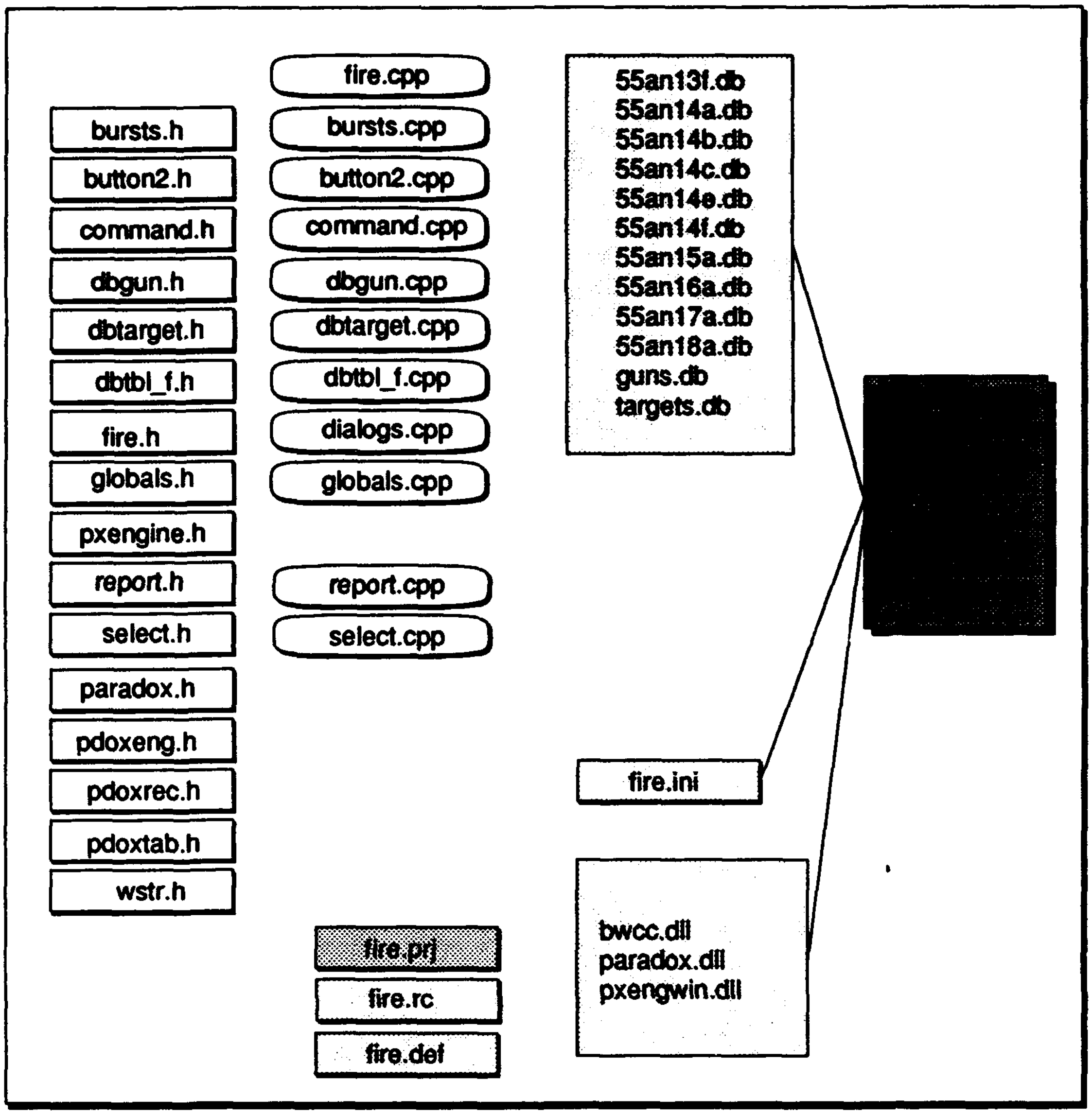


**Figure 8:** File organization of FATDS / FS (ofire)

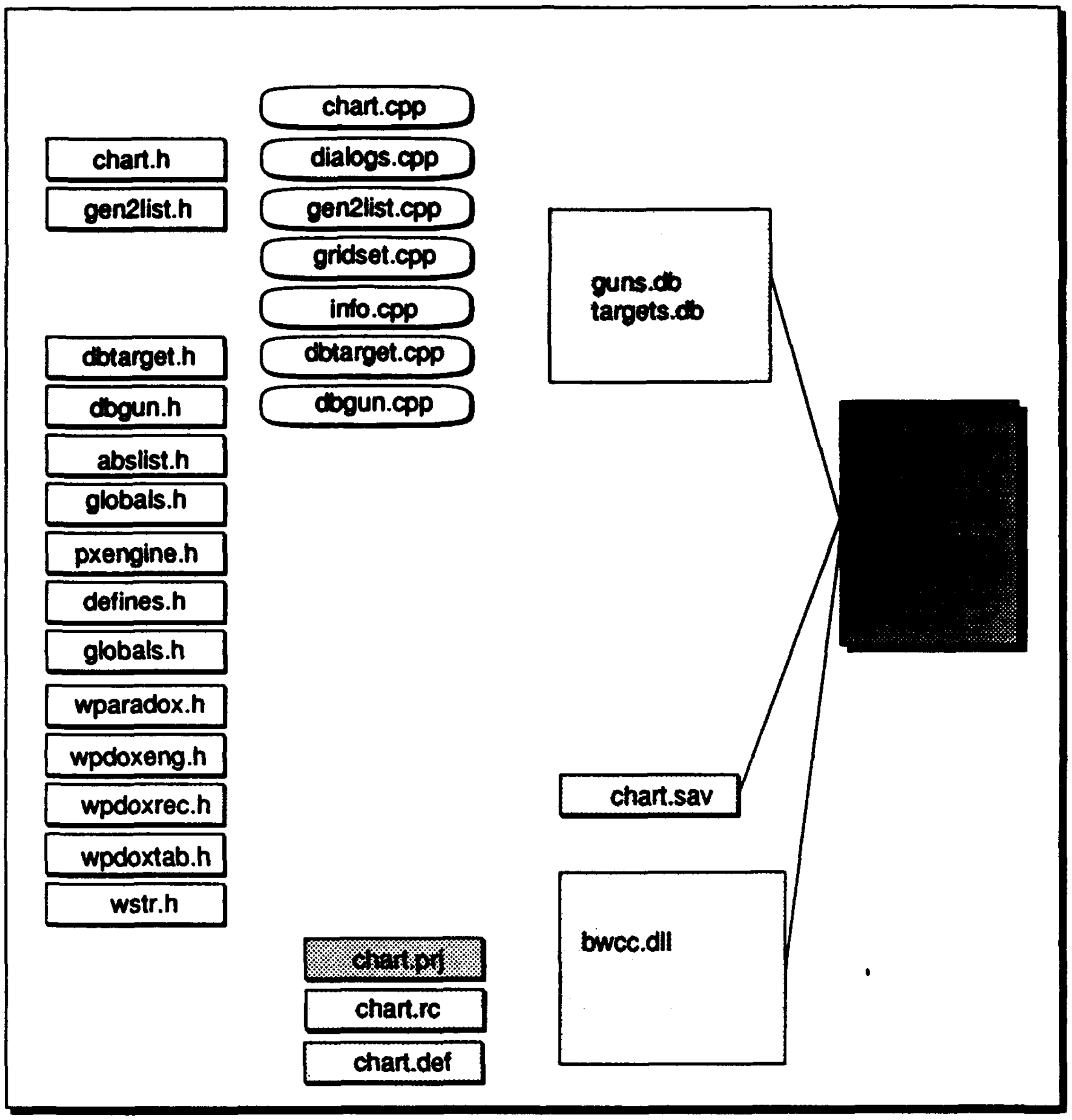


**Figure 9: File organization of FATDS / FS (ochart)**

## F. DESIGN DECISIONS FOR DLLs

Borland C++ and Windows both have limited number of DLL files. Each application adds new DLL files to the system. The more an application uses DLL files the more performance it provides to the user.

DLLs, necessary for out prototype are:

- Custom controls, e.g., button2.
- Database engine and other database service files which works with database engine.

## G. INITIALIZATION FILES

Some objects in the applications may need to initialize some of their attributes before they are created and store them back, just after they are deleted. The values of those attributes should be stored in a persistent environment. C++ does not provide a built-in database management system (it does not have to) to store such values. There are three options to do that:

- To use a separate DBMS
- To use binary files using C++'s stream libraries
- To use text files using C++'s stream libraries.

First option is not beneficial for such short amount of data. It increases the size of code and implementation time. Binary files store and retrieve the objects keeping their abstraction or their type.

For the OChart application, it runs using the last values of which just before it is closed. E.g., the user will probably want to open the firing chart with the same window size, at the same screen location, with the same scale and customization etc., as he/she close it for the last time. Each object stores these kinds of persistent data in a `*.sav` file. One file for each application is enough and it should be binary file.

Some persistent data which is useful for customization should be stored in text files with `*.ini` extensions. These kinds of data are mostly in string type or don't cost much to convert to/from other types. Its difference from `*.sav` files, `*.ini` files:

- Can be edited by the user without running the application using any kind of text editor.
- is useful to feed any type of control, like combo boxes and list boxes, with persistent data. Any object which can easily reach these data without worrying about data structure.

# IV. COMPARISON OF OBJECT-ORIENTED VS. CONVENTIONAL PROGRAMMING PERFORMANCES

## A. INTRODUCTION

As the need for the computers increases, so do the problem domain for the software. The software engineering goals, which are described in the first chapter, are sourced from the software problem. Some of the problems can be stated as [Ref.3: p.8]:

- Responsiveness: *Computer-based systems often do not meet user needs.*
- Reliability: *Software often fails.*
- Cost: *Software costs are seldom predictably and are often perceived as excessive.*
- Modifiability: *Software maintenance is complex, costly and error-prone.*
- Timeliness: *Software is often late and frequently delivered with less than-promised capability.*
- Transportability: *Software from one system is seldom used in another, even when similar functions are required.*
- Efficiency: *Software development efforts do not make optimal use of the resources involved (processing time and memory space).*

O-O is not a *silver bullet* to solve those problems, but it has important features which has more meaning in skillful hands. Following sections evaluates various criteria to put forward the compared performances of O-O against nonO-O programming environments.

## B. COMPARISON CRITERIAS

### 1. Modifiability

No software is perfect. There could be always errors. Even though a software meets requirements of the user, later the user will probably demand something more from that software. Modification is needed for two reasons:

- Errors
- New requirements on the software

Modification is almost similar to fixing or modifying a car. If the system is complex, modification should be located on the design job first. The role of design job or architecture of the system is to avoid getting lost in the complexity of the system. So the design should be consistent with the system. There are two step for modification:

- Locating the modification point(s) in the software system
- Performing the modification

First step is mostly effected from the design tools, while the second one from programming language environment. The keywords for both steps are *clarity* and *consistency.*

Windows programming environment provides some modification easiness of its own: Control tools (like buttons, string table, accelerations etc.) which can be observed from Borland C++ Resource Workshop, can be modified without even recompiling the whole software.

The relationship between major O-O features and modification will be discussed on next section.

### *a. Encapsulation*

Modification may come with some extra problems like:

- Increasing the complexity of the system unnecessarily
- Endangering integrity of data in the modules of the software

In C++, providing encapsulation, every object is responsible of its data and detail of its implementation. Mostly, objects hides their data from outer world, they just allow outer objects to use service or member functions to reach specific data of their own.

### *b. Inheritance*

Multiple inheritance is still in dispute about whether it is a strength or move of a “white elephant”, in O-O world. C++ supports multiple inheritance and should be used carefully, since it could be a big problem for modification. All the inheritance concept mentioned in this research is single inheritance.

Suppose that `FireCommandsWndCls` is a class and has member functions of `f1()`, `f2()`, `f3()`, and so on. After a time it become necessary to make some modifications: changing the implementation of `f1()`, adding a new behavior of `fnew()`.

In the conventional paradigms, modules, which does not reflects the user requirements, should be either:

- Thrown away, since they are not functional any more: *If a replacement is required, a new module should be created from scratch. Old code is wasted and becomes completely unfunctional.*
- Modified: *Software integrity should be preserved. Modified part of the code is lost.*

For the O-O paradigm, inheritance is another chance for *reusing* of the old code: Creating another class which inherits from `FireCommandWndCls`, overloading `f1()` and having `fnew()` member function. Big portion of the old code is still functional and the rest can still be used by others. Since the old code is preserved, its reliability is preserved too.

### c. *Polymorphism (dynamic binding)*

Consider the following sample codes derived from the OChart application:

A base class:

```
class SelectionToolCls
{
..
public:
..
   virtual void DoMouseDown(long, long);
   virtual void DoMouseMove(long, long);
..
};
```

Child class$_1$:

```
class DistanceToolCls : public SelectionToolCls
{
..
public:
   ..
   virtual void DoMouseDown(long, long);
   virtual void DoMouseMove(long, long);
   ..
};
```

Child class$_2$:

```
class NewToolCls : public SelectionToolCls
{
..
public:
   ..
   virtual void DoMouseDown(long, long);
   virtual void DoMouseMove(long, long);
   ..
};
```

`DistanceToolCls` and `NewToolCls` are two sample classes with their base class of `SelectionToolCls`. They all represent a button on the screen. If a user selects one of them then an appropriate action should take place. Those actions are the same by title but different by implementations. Since the interaction of the user can not be estimated in advance, conventional methodologies search for the user's interaction by `case`-like statements/functions. Each time the user's decision is wondered then it is searched. Adding and deleting buttons make the user to update all the `case` groups.

Dynamic linking or polymorphism makes modification easier and more reliable than conventional paradigms. Declaring a host pointer:

```
   SelectionToolCls *Selection;
   DistanceToolCls   DistanceTool;
   NewToolCls        NewTool;
```

And assigning sub-object to the pointer when needed:

```
   Selection = &Distance;
```

or

```
   Selection = NewTool;
```

Above assignment are made after user's interaction. Within the code:

```
   Selection->DoMouseMove(..);
```

or

```
   Selection->DoMouseDown(..);
```

The pointer `Selection` is independent from address object. It will be bounded dynamically and will retrieve appropriate class's virtual member function (which is last assigned one to the pointer). When a modification is needed, creating a new child class and making an assignment under a button control is sufficient. Polymorphism makes the code shorter and clear, so helps the modification.

### 2. Efficiency

The goal of efficiency implies that a software system should operate using the set of available resources in an optimal manner [Ref.3: p.30]. Resources are grouped into two:

- Time resources
- Space resources

The use of resources is mostly dependent on hardware and the algorithm which is used in the software. The O-O programming paradigm does not have significant impact on efficiency.

### 3. Reliability

The software system should operate for long periods of time without human intervention. This gains more importance especially for *mission-critical applications*. Software engineers accept that there is no perfect software. The goal to minimize the failures which endanger the mission. For increasing reliability, OOPLs provide:

- Object structure: *Provides realistic mapping from real-world objects to form the problem domain.*
- Encapsulation: *Protects the objects' own privacy and provide data integrity.*
- Reuse of reliable objects: *Assuming the old objects are well-tested, reusing of them via inheritance eases the testing of the new object(s).*

### 4. Understandability

Understandability is essential for modification and project management, because of the need for coordination. Understandability is dependent upon programming language and tools. Language is the way to express the real-world solution [Ref.3: p.31].

One of the fundamental differences, which distinguishes O-O from nonO-O is mapping from problem domain to software solutions. Conventional methodologies decompose a system into modules for which each module represent a major step in overall process. Object-oriented methodology keeps the real-world objects or decompositions in their abstraction and represent them in *objects*. "Call for Fire" process which is send from an observer to FDC for being a fire request, can be represented as an object in O-O. It has verbal operations and noun phrases, which describe its behaviors just like the real-world objects. It keeps its unity and communicates with other objects. The benefits of objects:

- Similarity of objects to the real-world objects makes disowning of O-O projects easier and more readable.
- Dealing with object modules is similar to real-world objects, so, maintains, modification and extensibility of the software is easier then conventional ones which are created by using top-down structured design.
- For large projects, it is easier to decompose the job without losing semantics of sub modules. This increases the reliability of subparts.

### 5. Code Size and Cost

One of the tools to estimate the cost of a software is Cocomo[1]. Cocomo is a lines-of-code-based costing model that estimates the software development cost. The cost is determined with the formula:

$$SM = A * KSLOC^{B}$$

In the equation, SM is the number of staff-months which is required to complete the system development; A is an empirically derived number that converts 1000 lines of code into its equivalent cost in SM; KSLOC is the lines of source code (expressed in thousands); and B is a factor that compensates for the nonlinear nature of software development. [Ref.19: p.363]

The role of O-O in decreasing the Line of Code (LOC) shows itself with its *reuse* capability. Reusing becomes more effective when:

1. Barry Boehm's Constructive Cost Model to estimate effort, man power, and so on [Ref.2].

- The size of code (or project) is too large (in ten thousands of LOC)
- Application completely or partially use high degree of reusable software components, like GUI component.

One point should not be disregarded, *reuse*, like cache memories, provides diverse performances on different problem domains and it mainly depends on *design*.

### 6. Portability

Portability is mostly dependent on both language and environment. O-O paradigm contributes to portability with its encapsulation feature.

FIGURE 10 on page 42 displays a sample Borland C++ include file which

```
#ifdef __BORLANDC__
//     PC-specific includes
#      include ..
..
#endif


#ifdef Unix
//     UNIX-specific includes
#      include ..
..
#endif
```

**Figure 10:** Selective includes of a sample Borland C++ header file.

behaves selectively for include files. Compiler checks the operating system and uses the appropriate include files. Since all the modules and objects have two parts which are interface and implementation, implementation part could have specific features to its environment. The User or programmer is not interested in that part. To port a software to another environment requires compiling & linking again.

### 7. Project Management

Designing and implementing systems consisting of tens of thousands, if not millions, of lines of code is quite different matter. The effort required to complete such a system is clearly beyond the intellectual physical capacity of one person. [Ref.3: p.7]

Gathering more people into a single project creates other problems: *Communication and coordination.*

FIGURE 11 on page 43 displays the resource allocation for O-O FATDS / FS in terms of time. The time which is spent for analysis and design is more than coding. This one of the characteristics of O-O. Design decisions effect the future works. *Reuse* should be a project goal like any other [Ref.16: p.44].

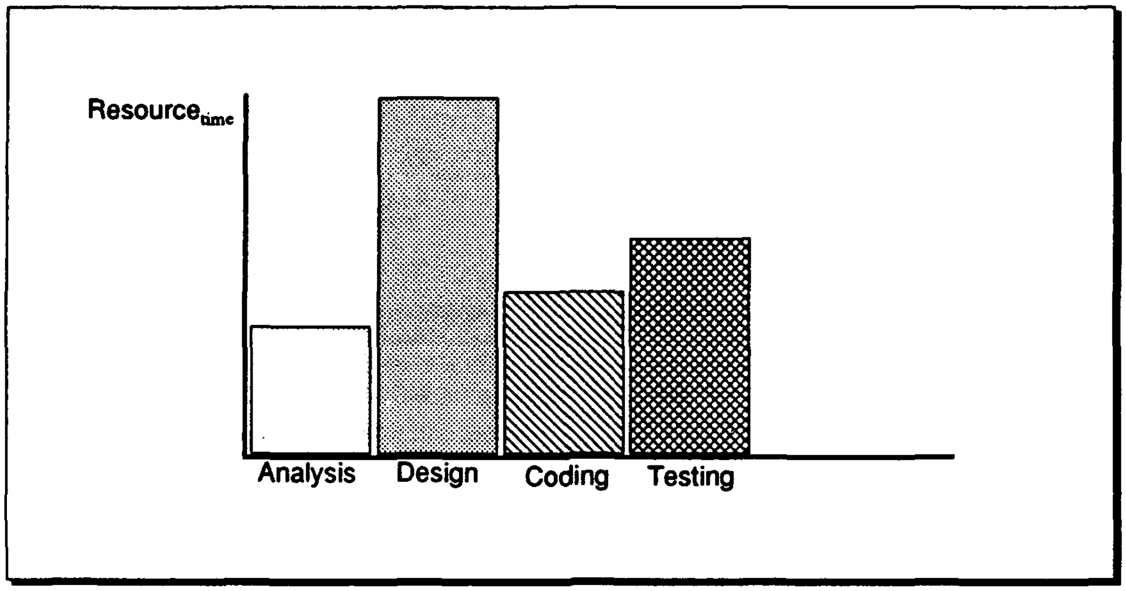


**Figure 11:** Time allocation for development elements.

### *a. Project Organization*

There are four types of alternative project development styles [Ref.16: p.44]:

- Rapid prototyping
- Round-trip
- The iterative style: *Develops a series of solutions to a problem, each one close to satisfying the requirements. Each solutions is complete but its accuracy or acceptability improves with each pass.*
- The incremental style: *Builds system functionality a little at a time. It differs from iterative development in that the results of each increment are not an entire solution, but part of it. This style has been advocated for some time for conventional projects.*

Object-oriented development is best suited to the iterative or incremental style of development. Abstraction of the real world objects in O-O terms makes possible to work with objects or modules independently. Object can be put in action as soon as it possible whether they are completely ready, or not. This gives a chance for:

- Early evaluation of objects which justifies the rest of the work.
- Estimating the cost of the remaining job more realistically.
- Allocating the resources equally for mid-term goals.

For is research a model named *model-based project development cycle*, has been applied for FATDS/FS. It is both incremental and iterative. FIGURE 12 on page 44 shows the object development algorithm with MBPDC. OOD sets the goals of each object. Since an object may not be fully functional at moment during the implementation-testing process, mid-term expectations should be planned for them (current goal(s)).

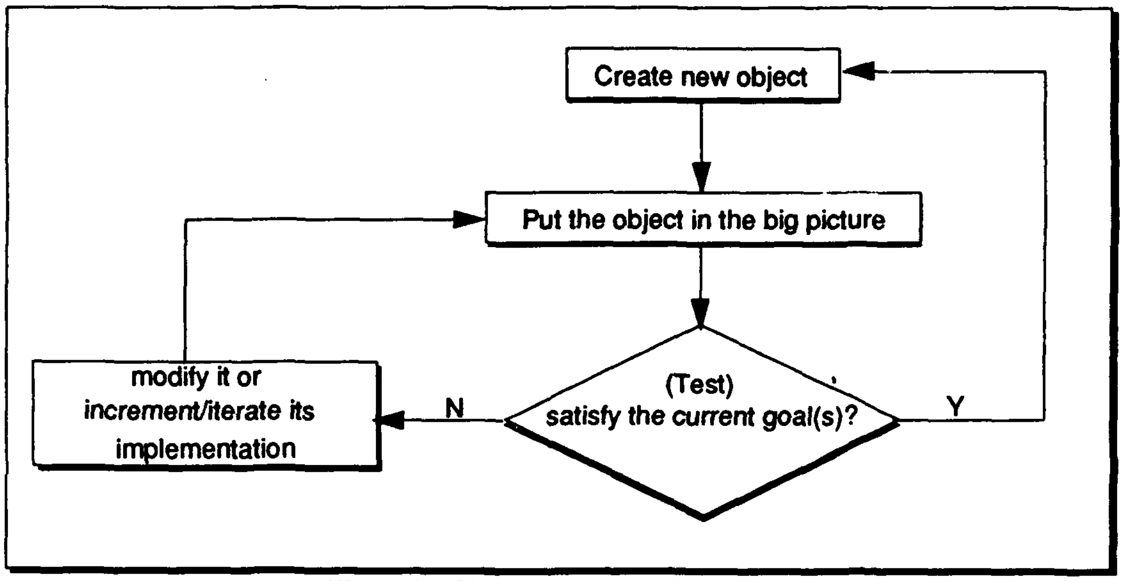


Figure 12: Object development in MBPDC.

Conventional design methodologies have the problem of decomposing the problem domain in to functional sub units. Since their modules represent verbal actions, conventional languages are not good for iterative development cycle.

# V. DATABASE MANAGEMENT SYSTEM (DBMS) SELECTION FOR FATDS

## A. INTRODUCTION

FATDS has a great deal of data to manipulate. Data should be:

- Kept in reliable environment(s)
- Shared from other nodes of the system

FATDS's database system should provide:

- A distributed environment: *Supporting multiple workstations connected to a local or wide-area network, with one or more database servers providing transparent access to multiple data sources.*
- An advanced user interface: *Provided by a graphics workstation or PC with the capability of integrating text, graphics, pictures, and possibly sound and video.*

Before implementing the database part of the project two questions should be answered; First, what will be the DBMS's source? It could be:

- Programming language's own DBMS.
- Application Developers's own DBMS.
- Commercially available DBMS.

Second, what will be the database model for DBMS? It could be:

- Object-oriented database model.
- Other database models.
- Composition of database models.

## B. CHOOSING A DBMS SOURCE

### 1. Programming language's own DBMS

Borland C++ does not have a built-in DBMS like most of the other programming languages. If the programming language has provided a DBMS, then:

- It would be completely object-oriented.
- It may be easy to embed DBMS functions or objects into the source code.
- It may not be satisfactory for the applications requirements, e.g., security, integrity, suitability to distributed databases and/or for large objects.

- There would still be problems about using existing databases (which have been created by other DBMSs.
- It would have enforce to use the same language's native DBMS format for other applications.

### 2. Application Developers's own DBMS

Another option is to develop a DBMS which has been created specially for the project or general purpose. It is not commercially available. If it is task-oriented then it meet all the requirements of the project and acts well with the application code. It does not have to be implemented with application's language, but to embed database system calls there should be third party communication programs in application's language.

One of the features of this implementation is that it is completely object-oriented like the first option. Since this choice under the control of the application developer, it can be maintained or modified according to requirements.

The problem with these kind of DBMSs is maintenance. Its development should be included into the project, hence, this increases the project burden.

### 3. Commercially available DBMS

Commercially available DBMSs are also for general purpose, however there is a selection option according to the project requirements. These DBMSs are developed outside the project (FATDS) environment, so, developing a DBMS never be another task, but to use it. It is possible to use/create various databases other than the project's. If it is a widely used commercial DBMS, it will possibly be used longer.

Disadvantage with this choice is maintance. Only the DBMS vendor can provide necessary modifications on the application.

### 4. Choosing the DBMS Source for FATDS

Using a commercially available DBMS is chosen for FATDS. If it does not specifically violates the requirements, "buying a product is generally cheaper then producing it".

One of the advantages of commercial DBMSs is that database can be created without using the host application. This provides to use the same database from other type of applications locally or via network.

## C. CHOOSING A DATABASE MODEL

### 1. Database Model Options

There are various data models. The major data models are ***hierarchical*, *network*,** *relational* and *object data models*. Each of the data model is not alternative of the other(s). There are advantages, disadvantages and suitable applications for each the data models.

In *hierarchical data model*, data is organized in inverted tree structure and is accessed form the top the bottom in a series of nodes similar to branches in a tree. The inverted tree structure provides clearly defined and fixed path for accessing the data by traversing the branches of the tree. The rigidity of the structure has some disadvantages in that new access path, not defined at he outset, may complicated or even impossible to achieve. Modification of the database structure is a very complex task.

*The network data model* traded off some of the high-access performance for flexibility in organizing and accessing the data. As in a hierarchical model, the data is organized in and inverted tree. The greater flexibility is achieved by allowing a node to be connected to more than one branch. Records are linked by predefined pointers. Equal or better performance, as compared to the hierarchial model, was achieved by maintaining permanent pointers linking records. The complexity of the data structure made queries more complex and modification and ad hoc queries more complicated

The *relational data model* promised a structure that met these new requirements. Structurally different from inverted trees of the hierarchical and network data models, data in the relational data model is stored in tables consisting of columns and rows. Links between data records are largely established as needed on an ad hoc basis. The rows of a table represent records and the columns within a table represent the fields in a record. Different types of records are stored in separate tables. Some frequently used connections between tables can predefined, while others can be established at the time of query. The data and their description are maintained separately, but changing data description is not simple. This model achieved the desired flexibility and ease of use, but at a cost. The relational data model requires a higher level of processing to establish connections and access data. Consequently, system requirements for processor and memory are higher to achieve the same level of overall performance.

The *Object data model* differs itself from other models representing real-world data in intelligent objects which combines the abstraction and behavior of the real-world objects. Direct correspondence between real-world and database objects maintains integrity and identity of objects so they can be easily identified and operated upon [Ref.13: p.442]. Objects have their own behaviors, this provides them to interact with other data models. Advantages of the object data model: [Ref.13: p.444]

- Extensibility: *Object types and their methods can be modified as needed. Such changes are localized to the relevant object type and hence are much easier than in record-based system.*
- Behavioral constraints: *Because of encapsulation, the behavior of each object type is predetermined by a fixed set of methods. Hence, database operations are constrained to be within these behavioral specifications.*
- Flexibility of type definition: *The user is not limited to the modeling concepts of data model but can define many variations of data types, each with unique properties.*

- Modeling power: *The inheritance of both attributes and methods is very powerful tool for data modeling. In general, the abstractions of generalizations/specialization, identification, and aggregation are well supported in the current data models.*

Disadvantages of the object data model:

- Lack of associations: *The abstraction of association is not directly supported and is achieved indirectly by allowing interobject references. This is an inherent weakness of the O-O approach.*
- Behavioral rigidity: *The predetermining and prespecifying all operations by a fixed set of method create this rigidity.*
- No high-level query language: *There are no high-level query languages for current data models.*

## 2. Interfacing Objects with the Other Models

Object-oriented databases provides to write and read objects to the disk, normally using static or dynamic link libraries with the code.The physical storage format on the disk is hidden from the application. The format can be flat files, relational files, hierarchical files, or any other type.

The most primitive OODBMSs use a custom file format and enable you to only store objects on disk. These products offer little advantage.... Likewise, the manufacturers of these products require you to purchase their software and, occasionally, a run-time license for each product distributed. [Ref.20: p.5]

A OODBM may either:

- Store its objects using a format compatible with existing commercial database management systems such as Oracle, Sybase, dBase IV, or Paradox: *Data consistency is ensured and all transactions are performed internally by the OODB. The disadvantages are technical difficulties and the relatively slow performance.*
- Exists in parallel with a commercial database management system that mirrors the data in the object-oriented database: *It requires an update scheme to ensure that the data in both databases remains in synch. Accessing data in an object-oriented database is much faster -2 to 10 times faster- than in an relational database.*

## 3. Choosing a Data Model for FATDS

There is no single data model which completely fits FATDS's database requirements. Object-oriented databases can interface with other data model. This provides the opportunity of combining the advantages of different data models. For this research, interfacing objects with relational DBMS is selected.

In relational DBMSs, the structured organization of data in tables and the capability of the database system to interpret each column of data serves well for applications involving pure numerical and textual data. Variable length data elements such as free-form text, images, and voice does not fit into the structure of relational databases. Relational database vendors created a new class of data type called Binary Large Objects (BLOBs). These data types can hold very large (and variable length) data and they are not interpreted by the database. This new feature provides relational DBMSs to support multimedia data management.

Relational database model has advantages that the OODBM can not support: [Ref.22: p.68][Ref.26: p.33]

- The relational schema is stored in the database catalog.
- General-purpose query programs can use the catalog.
- The SELECT, PROJECT, and JOIN operators can build new database views at run time.
- The database is cross-compatible with other applications that use the same DBMS.
- The data and their description are maintained separately.

The object data model has advantages that the relational data model can not support [Ref.22: p.68][Ref.18: p.169]:

- Variable length data member: *It can support such applications as imaginary, multimedia, geographic data, weather.*
- Abstraction: *The behavior of an object is described by a class definition that is created by data abstraction. By incorporating data abstractions at the level of the database, it is possible to make changes to the way a database class is implemented.*
- Class extensibility: *In a relational database, the only parameterizable type is a **relation**, and the only operations possible on all relations are **get_field_value** and **set_field_value**. In the object data model, interface of each object is customized to that object.*
- Arrays permit repeating groups.

- Encapsulation: *Encapsulation of data formats with methods bind data representation and behavior.*
- Polymorphism: *Customizes the behavior of derived data types.*
- Integrity: *Especially for large systems, the integrity of* ***analysis, design*** *and* ***programming*** *to map the real-world data into objects provides the simplicity to build the database system. This provides easy maintenance of the database system.*
- Reusability and adaptability: *They are the two important features of object-oriented systems, which are accomplished using inheritance.*
- OODBMSs are faster: *Because, relational databases pull information from a variety of tables into one result set, based on JOIN operation while OODBMSs make the same job by calling member functions.*
- Automatic type checking at the point of use: *The object-oriented database performs type checking on the arguments to method calls. Thus, type errors are detected at the time of invocation rather than at the end of a transaction.*
- Schema evolution: *Conventional data models do not support efficient mechanisms for schema evoluton. Object data model allows user-defined operations making the addition easier. Schema evolution includes changes to the defination inside a class and changes to the structure of the class lattice. A set of invariants and rules can be defined so that the schema will remain consistent as it evolves.*

## D. DBMS FOR FATDS

FATDS's database is implemented in object-oriented fashion which interfaces a commercial DBMS, Paradox. Borland's Paradox for Windows, version 1.0, is a relational DBMS. Paradox provides network support, password protection using encryption, binary large objects (BLOBs), multiple indexes, and composite indexes.

To use Paradox table in the applications and embed into the C++ source code a third party Engine is need: Paradox Engine. The Paradox Engine, version 3.0, is and API containing more than 90 functions accessible from a C, C++, or Pascal program. The Engine provides to create database tables and indexes and to perform all database-related operations. The Paradox Engine is implemented as a DLL (for windows) that can be distributed royalty-free. The problem with the Engine is the DLL file is implemented in C, so it is not object-oriented. To keep the applications completely implemented in O-O,

**another DLL file is created, which is in O-O. The design and implementation issues of FATDS's database is discussed in the next chapter.**

**The benefits or advantages of this choice:**

- Working with other data format files too: *The application(s) may have to work with other data formats. FIGURE 13 on page 53 displays the object-interfaced data*

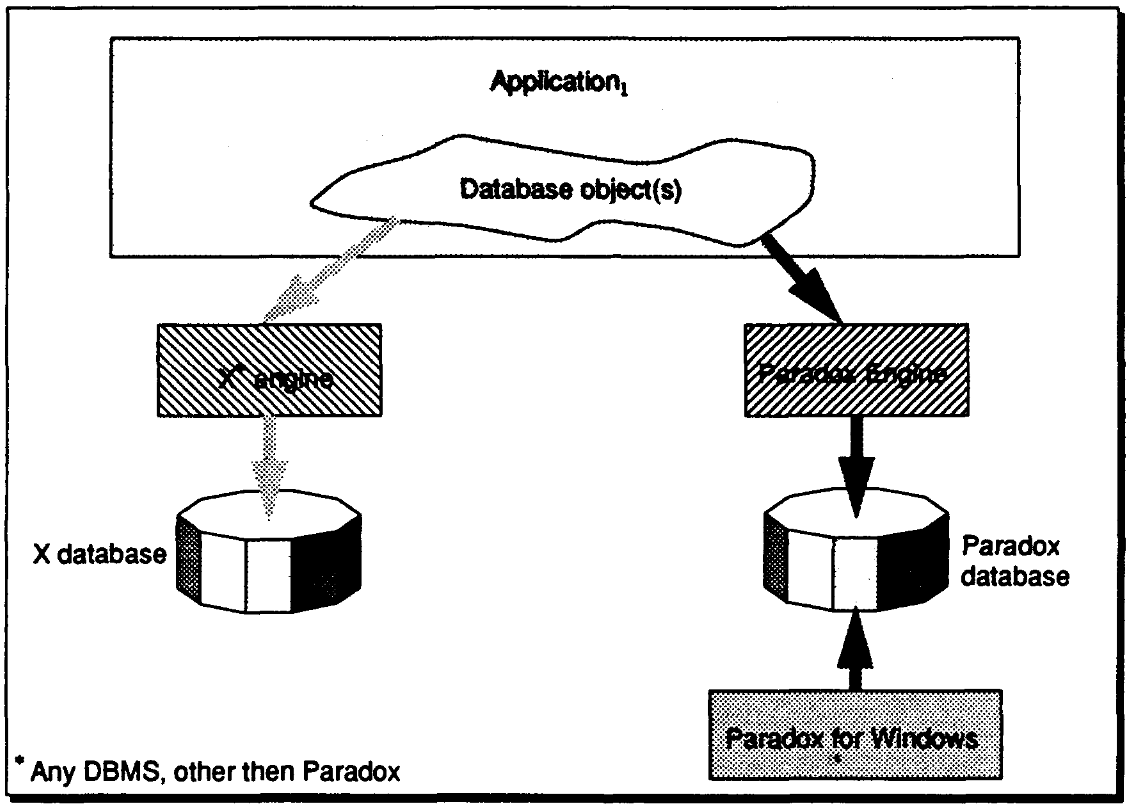


**Figure 13: Object-interfaced data environment**

*environment. Objects can be implemented to filter diverse database formats, hiding unnecessary details from the users and end-programmers. Only the database object are necessary to be updated according to the target database, not the application. This makes the modification easier.*

- Using a reliable DBMS: *Instead of creating a DBMS which is normally a independent task, using a commercially developed (so tested0 provides more reliable DBMS environment.*
- Faster database development: *Data does not have be created or maintained by the mission-oriented applications. It should be created/maintained without running the application or creating a special program. Database design and implementation of*

*FATDS, including custom forms, can be created by Paradox for Windows, without using any user-created application.*

- Using the data with other applications: *Other applications created independent from FATDS may use the same data. Regarding the hugeness of the project and time period of using the application, generally accepted data formats provides flexibility in use.*

# VI. OBJECT-ORIENTED DATABASE DESIGN OF FATDS

## A. INTRODUCTION

The database system of FATDS is determined as object-oriented database system which incorporates a relational DBMS. OODBMS has a front-end role in this system and back-end could be any DBMS in any data model. Object structure of the system facilitates to work with heterogenous back-ends. Using various DBMSs as a back-end effects the design of the database. Back-end database could be:

- Created independent from the front-end implementation.
- Created simultaneously or incorporating with the front-end.

Two of the situations may effect the object design differently. Creating the object-database as a front-end together with the back-end database may increase the reusability of the objects for later developments. For this research first option is followed to show:

- How object-oriented database systems coexists with the existing databases.
- The benefits of object-oriented database system interfacing with relational DBMSs.

One of the benefits of the of OODBMSs is their implementation language. Non-O-O database system mostly have their own data languages which are not capable of complete computational power like programming languages. OODBMSs' languages are complete programming languages as well. This facilitates to combine database development and program development together.

Design methodology will follow the steps:

- Identify the basis for the database requirements
- Define the database's functional and performance requirements
- Conceptual design
- Identify classes
- Identify attributes and services (member functions)
- Identify the relationship between classes
- Database implementation

## B. DATABASE REQUIREMENTS AND ANALYSIS OF FATDS

### 1. The Role of Artillery

The mission of field artillery (FA) is to destroy, neutralize, or suppress the enemy by cannon, rocket and missile fires and to assist in integrating fire support into combined arms operations. The main role of FA is to ease the achievement of commander's objective with its fire power. The commander is a maneuver unit commander at any level.

Targets could be anything on the battlefield: Enemy units, bridges, areas, roads, tanks, trenches, convoys, etc. Main point is that targets are not artillery's, but the commander's. So, all the target acquisition elements work under the maneuver commanders' control.

Since artillery normally does not have eye contact with the enemy targets, he relies on different sources for the target information; forward observers, air observers, radars, electronic counter measure units, higher headquarters, reconnaissance/intelligence units, commander himself, etc., almost everybody on the battlefield is source for target acquisition. A target could be immediate one or planned before.

Depending on the organization of the FA there could be diverse weapons systems. Each weapon system has its own munitions and firing tables. Firing table is a book which displays all numeric data about a specific gun & munition(s) combination.

Although there are many, only the following major application functions are taken into account for database implementation:

- Target management
- Gun management
- Firing table management (Table-F only)

Other database areas are:

- Corrections management: *It keeps and manipulates firing corrections for each gun and position pairs.*
- Fired mission history management: *It is a derived database management.*
- Known points management: *Radars, observation post positions, etc., are all known points.*
- Ammunition management: *Amounts, type, caliber, lot are the fields to be managed. (It is designed but not implemented.)*
- Met messages management: *It keeps up-to-date met messages.*
- Friendly forces management: *Positions of friendly forces and other regions which are necessary for fire coordination, planning and security.*

### 2. Application Functional Requirements Overview

#### *a. Target Management*

Each target is assigned a number like "AB2124". First three digits represent a specific unit or element while the rest represent consecutively target numbers which are assigned to that unit/element. All the targets should be located at least with its coordinates, height and description. In addition to those: source (who located), locating accuracy, located time, category (there are 13 category groups and each target fits at least one of them), fired/notFired, priority, restrictions.

Targets could be a point, a region (with a size), target groups (two or more targets with a different identification name), target series (more than one targets and/or target groups with a special name). Battalion FDC can form a target groups and series. A sample group name is B1C; letters represents who forms that group, the number digit in the middle is consecutive group number. A sample name for series of targets is CASEY, just a nick name.

The decisions that comrise the decide function include:

- What targets should be acquired and attacked.
- Where and when the targets will likely be found and who can locate them.
- How the targets should be attacked.
- Whether target damage assesment is required.

FDCs will be able to keep track of unlimited number of targets.

### *b. Gun management*

Each gun has a unique number within a battery. Together with battalion name and battery name each gun has a unique name. Guns can be organized in some batteries as platoons (four guns in a platoon and two platoon in a battery), if not, a battery has six guns. All the guns in battery are identical to each other. One of the guns in the battery is assigned as the main gun.

In addition to name and each gun has position coordinates, model, maximum range, maximum left/right deflection, common orientation deflection, orientation deflection, firing table name.

### *c. Firing Table Management*

Each gun has its own firing table. A firing table is composed of various tables:

- Table-A: Line numbers of meteorological message.
- Table-B: Complementary rang line number.
- Table-C: Wind components.
- Table-D: Temperature and density corrections.
- Table-E: Propellant temperature.
- Table-F: Basic data and correction factors.
- Table-G: Supplementary data.
- Table-H: Rotation-range.
- Table-I: Rotation-azimuth.
- Table-J: Fuze correction factors.

Above tables are for specific shell type/model, fuze type/model and propelling charge/type.

All the firing table data are constant. Any application should not be allowed to modify the tables.

## C. DATABASE DESIGN OF FATDS

### 1. Conceptual Design

FIGURE 14 on page 60 and FIGURE 15 on page 61 display the ER diagrams for the Target, Gun and Table-F entities. The conceptual design is performed according to relational data model concepts.

### 2. Identify Classes, Attributes and Services

The classes which are formed as the front-end database classes are:

- DBGunCls: *Gun database management class.*
- DBTargetCls: *Target database management class.*
- DBTable_F_Cls: *Table-F management class.*

Tables 3, 4, 5 displays the data members and data functions of the database classes:

TABLE 3: DATA MEMBERS WHICH ARE USED BY ALL THE DATABASE CLASSES.

| Name | Access | Description |
|---|---|---|
| `nFields` | Private | Holds number of the fields in the database table. |
| `Names` | Private | Holds the names of the all the table field names. |
| `Types` | Private | Holds the type codes for each field of the database table. |

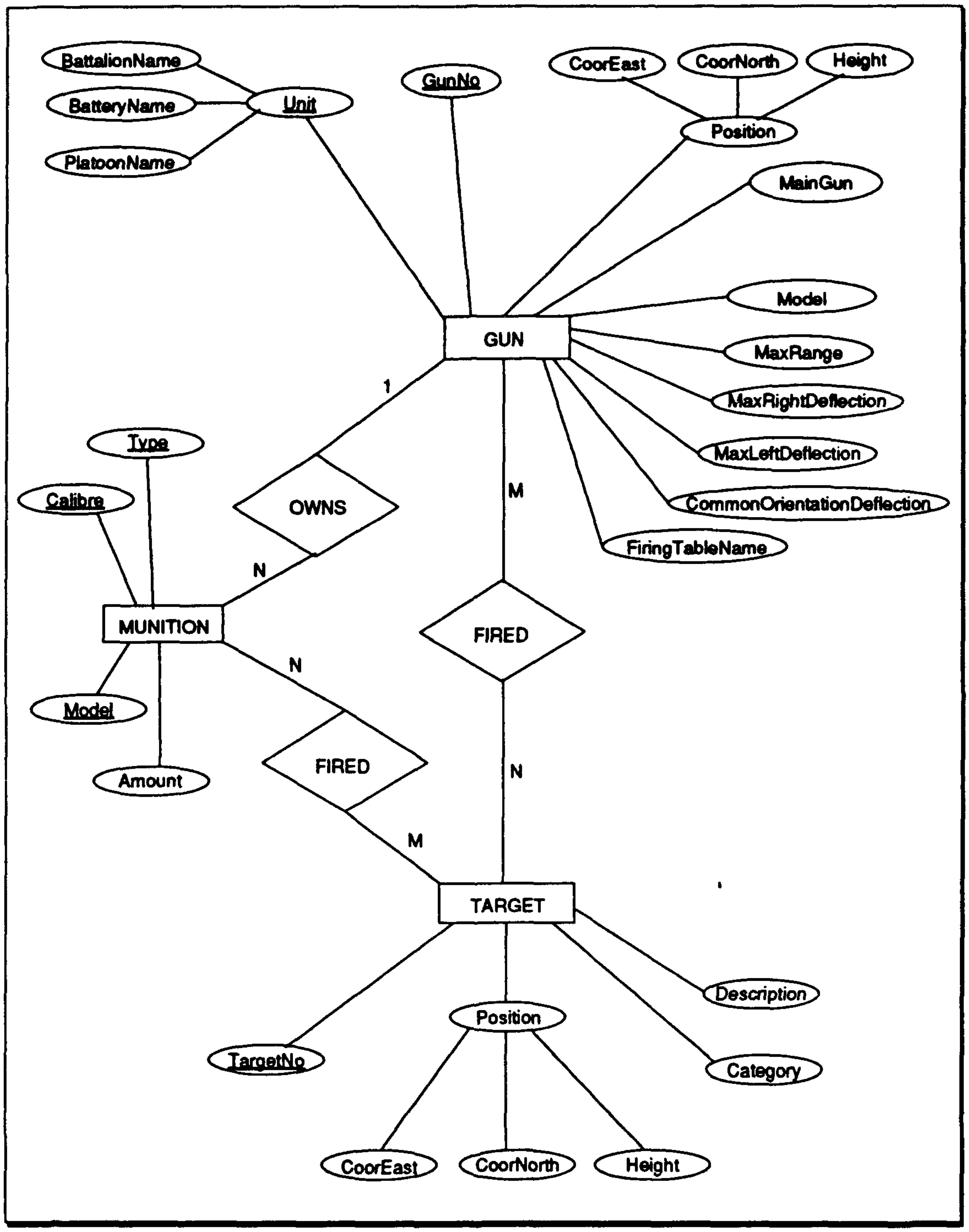


**Figure 14:** ER diagram for FATDS/FS schema (gun, target, munition)

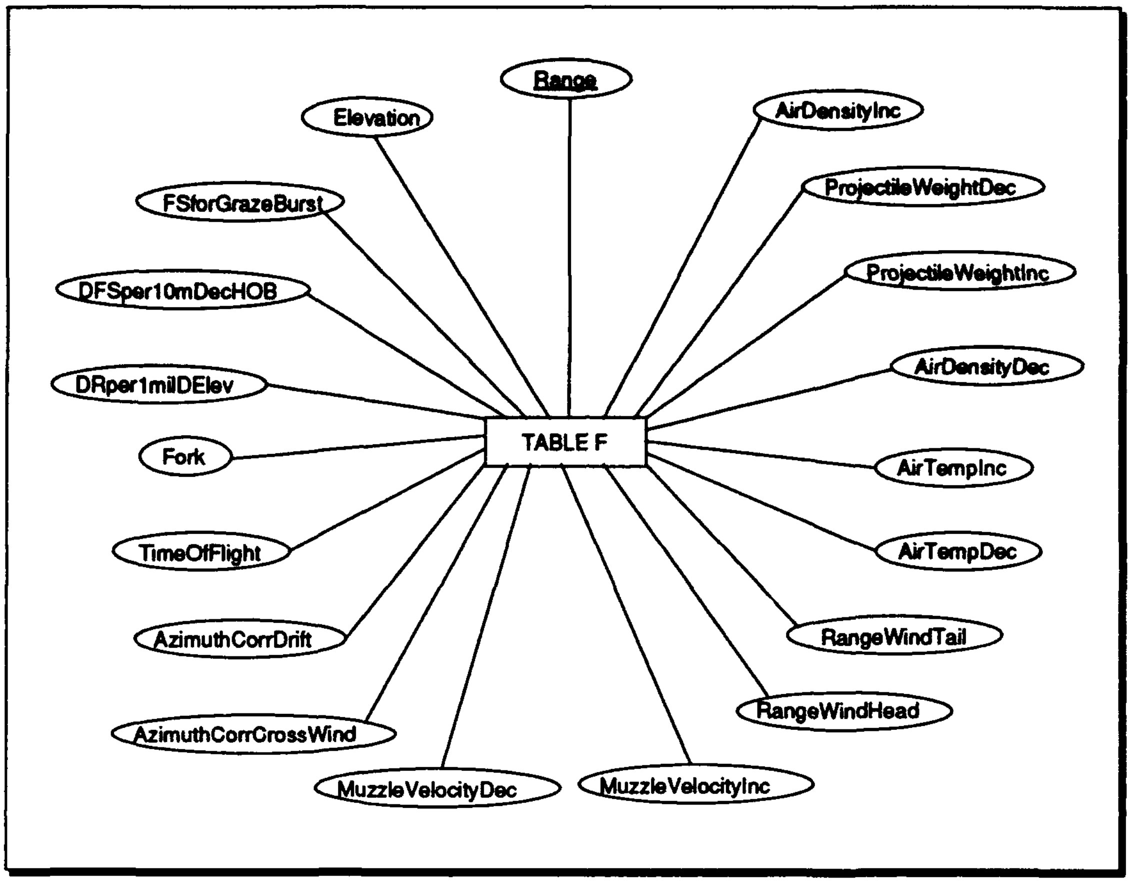


**Figure 15:** ER diagram for FATDS/FS schema (Table F)

TABLE 4: MEMBER FUNCTION WHICH ARE USED BY ALL OF THE DATABASE CLASSES.

| Name | Access | Description |
|---|---|---|
| `RecordTobuffer` | Protected | Takes the field values from the database table from where the pointer is, and puts it to the buffer fields. |
| `ReadFirst` | Public | Moves the database pointer to first record and reads the record into the record buffer. |
| `ReadLast` | Public | Moves the database pointer to last record and reads the record into the record buffer. |
| `ReadNext` | Public | Moves the database pointer to next record and reads the record into the record buffer. |
| `ReadPrev` | Public | Moves the database pointer to previous record and reads the record into the record buffer. |
| `isTableEmpty` | Public | Checks the table if it exist or empty and return TRUE if it is empty. |
| `isLast` | Public | Checks the if the pointer is at the last record and returns TRUE if it is the last record. |
| `Reset` | Public | Empties the record buffer and sets all the attributes to initial states. |

TABLE 5: MEMBER FUNCTION WHICH ARE MOSTLY USED BY SOME OF THE DATABASE CLASSES.

| Name | Access | Description |
|---|---|---|
| `BufferToRecord` | Protected | Takes the values from the buffer and puts them to the database table to where the pointer is. |
| `AddRecord` | Public | Adds the record buffer to the database. (The record is appended if the database does not have a primary key that requires the database to be kept in sorted order.) |
| `UpdateRecord` | Public | Copies the record buffer into the database as an update of the current record. |
| `DeleteRecord` | Public | Deletes the current record. |
| `SearchByKey` | Public | Searches the key field for a given field value. |
| `Set*` | Public | Set the a field value into the buffer area. |
| `Get*` | Public | Retrieves a field value from the buffer area. |
| Any other specific member function(s) | | |

## 3. Database Implementation

### *a. Paradox Database Tables Organization*

Table 6 displays the valid field types used in Paradox tables.

TABLE 6: PARADOX FIELD TYPES

| Type | Storage Size in Bytes | Description |
|---|---|---|
| N | 8 | Floating point number with 15 significant digits in the range of $10^{-307}$ to $10^{308}$. |
| S | 2 | Integer in the range of -32,767 to 32,767. |
| $ | 8 | Same as floating point, but fixed at two digits of the decimal. |
| D | 4 | The date. |
| Annn | 1 to 255 | NULL-terminated string of length nnn, where nnn is less than 255. |
| Mn | n+10, where is in the range of 0 to 240 | Unformated text BLOB. n represents BLOB-related header information, and the actual data is stored in a separate file pointed to by the file name stored in Paradox. |
| Bn | n+10, where is in the range of 0 to 240 | Same as the preceding, but for unformated binary information. |
| Fn | n+10, where is in the range of 0 to 240 | Same as the preceding, but for formated text. |
| On | n+10, where is in the range of 0 to 240 | Same as the preceding, nut for Windows OLE objects. |
| Gn | n+10, where is in the range of 0 to 240 | Same as the preceding, but for graphics data. |

### *b. Database File Organization*

The Paradox Engine provides the *pxengwin.dll* DLL file together with *pxengine.h* header file. Engine is implemented with C language, thus it is not object-oriented. The Engine with its more than 90 functions provides all the database file manipulations including:

- Create, read, and write Paradox tables, record, and fields in DOS, Windows, and network environments.

- Create, read, and write Paradox BLOB fields.
- Support explicit interactive Paradox and Paradox Applications Language (PAL) applications, as well as other Paradox Engine Applications.
- Support other Paradox features such as password protection, encrypted table, daie encoding, searching, and error handling.
- Import data into Paradox tables through serial communications, such as when downloading form mainframes or from external devices that can not be reached from PAL.
- Create stand-alone applications or applications that can be run with the PAL RUN command.

FIGURE 16 on page 66 displays the file organization for the database development of FATDS. In order to create an object-oriented database engine, another DLL file is created namely *paradox.dll.* Paradox.dll is created with the files:

- Paradox.*: *Loads other header files required for others, support proper memory allocation if multiple applications are simultaneously using the paradox.dll.*
- Pdoxeng.*: *Initialize the engine and shuts it down.*
- Pdoxrec.*: *Performs single record operations in a database table such as putting data into the field or get data from the field.*
- Pdoxtab.*: *It the engine is initialized it attempts to open a specific database table. If there is not that table, then it creates new one.*

Paradox.dll uses the Paradox Engine only. Database classes work with Paradox.dll. This hides all the Paradox implementations into the database objects. Application(s) is just aware of an object-oriented database implementation as a front-end. The database objects may interface any DBMS.

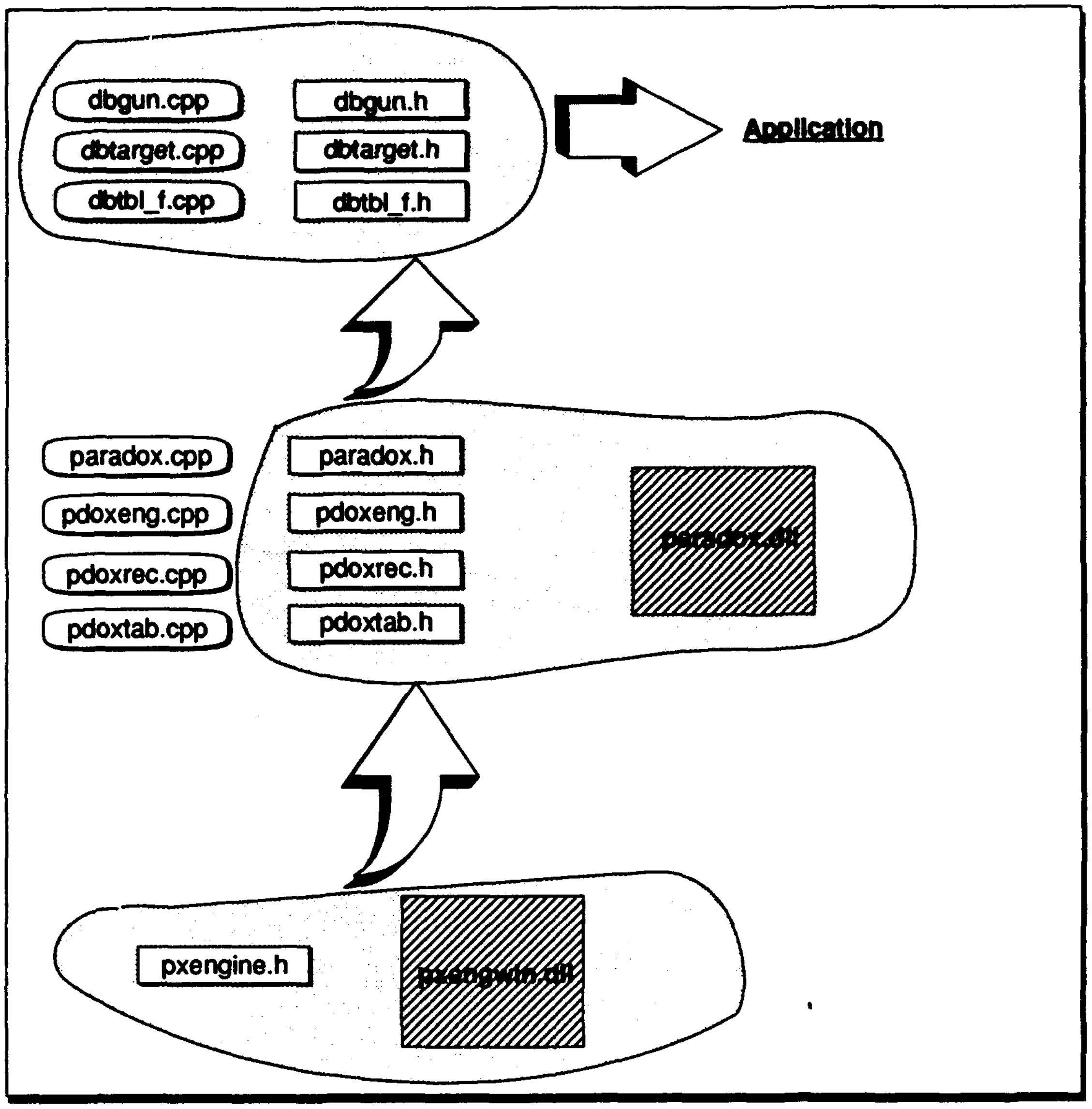


**Figure 16:** Database development file organization

# VII. CONCLUSION AND FUTURE WORKS ON FATDS

## A. SUMMARY AND CONCLUSIONS

This thesis concentrated on the benefits of object-oriented approach over FATDS on sample programming implementations. The object-orientated approach mainly distinguishes itself form the other conventional ones especially in two phases: *Design* and *maintenance*. Object-oriented approach offers increased modeling power to represent the real world in a problem domain, a high level of abstraction, and the ability to inherit and refine object properties. Design takes the greatest portion of the software development cycle. Maintenance is not just an *after-development-event*, but a continuing phase with the design during the software's life-time as well.

The object-oriented development of FATDS is accomplished using a methodology namely *model-based project development cycle*. This methodology depends on the fact that nothing is perfect and a question that is "is this what you want?". A development of a project should begin somehow, with a model of the product. The model is a prototype-like working element which gives an idea about the product but not necessarily a completely functional one. This model provides a chance to evaluation about the product and justifies the rest of the work. Evaluation causes modifications on the project. The inheritance, polymorphism and object abstraction features of object-oriented approaches facilitates the modification of the software and provides faster project development which satisfies the requirements than the conventional paradigms.

By promoting code reusability the object-oriented paradigm reduces the cost and time required for software development. In order to exploit the reuse and portability of power of O-O approach, design plays the key role. Object-oriented approach does not differentiate itself from other approaches without a good design. Thus, project members need to be experienced on OOA and OOD.

FATDS has the following features which make to work with object-oriented approach more beneficial than of the conventional ones:

- Large-scale systems.
- Modification on the software is frequently needed with new concepts, weapons, communication systems etc.
- FATDS operators are short-termed trained on that system(s).
- Incorporates a diverse large and distributed database.

At the beginning of a object-oriented software development, the power of reuse may not put itself strong since there may not be much objects which are created before. As the development goes on, the speed of project development increases logarithmically.

Advantages of object-oriented databases are that it offers the designer a high level of flexibility and power to implement arbitrarily complex and operations.

Object-oriented database structures should resemble C++ or some other object-oriented language program structures for data in memory (that is, the equivalent of class instances or objects), thereby minimizing discontinues between programming and database operations and structures. A good OODBMS must be a full object-oriented database system that is designed to support large-scale object databases. It should reflect the object paradigm with object encapsulation and inheritance of both data and functions.[Ref.1: p.169]

## B. FUTURE WORK

The programs, both the *OChart* and *OFire* are not completed and well-tested for complete functionality, so is the database of them. The followings will be done on the programs:

- Firing with special munitions
- Firing with met + MV.
- Interfacing with outer devices.
- Creating the other parts of the firing table databases and for other guns as well.

To continue on programming development increases the objects which are created, so does the reusability. This gives more reliable bases to compare object-oriented paradigm with the others.

# APPENDIX A. (OFIRE HEADER FILES LISTINGS)

## Listing A.1: BURSTS.H header file

```
//*********************************************************************
//
//                              bursts.h
//
// header file for the classes BurstsWndCls
//                              SheafWndCls
//*********************************************************************


#ifndef    __BURSTS_H
#   define __BURSTS_H

//note that the following code is executed only
//if this file has not been previously included



#include "fire.h"



#define ID_CONVERGED 101
#define ID_PARALLEL  102
#define ID_SPECIAL   103
#define ID_OPEN      104
#define ID_GUNS      105



_CLASSDEF(SheafWndCls)
_CLASSDEF(BurstsWndCls)
//=====================================================================
// class                      BurstsWndCls
//
// Purpose   : Displays the bursts on a canvas for each gun and allows
//             the user to modify them.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//=====================================================================
class BurstsWndCls : public TWindow
{
private:

   RECT         WndRect;
   PSheafWndCls SheafWnd;
   PButton2Cls  ButtonConverged;
   PButton2Cls  ButtonParallel;
   PButton2Cls  ButtonSpecial;
   PButton2Cls  ButtonOpen;
   HBITMAP      hBitmap1;//north sign
   HBITMAP      hBmpLeftArrow;
public:

   PTListBox    ListGuns;

   BurstsWndCls(PTWindowsObject AParent,
                LPSTR           ATitle,
                int X, int Y, int dX, int dY);
   ~BurstsWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void ConvergedSheaf(RTMessage) = [ID_FIRST + ID_CONVERGED];
   virtual void ParallelSheaf(RTMessage)  = [ID_FIRST + ID_PARALLEL];
   virtual void SpecialSheaf(RTMessage)   = [ID_FIRST + ID_SPECIAL];
   virtual void OpenSheaf(RTMessage)      = [ID_FIRST + ID_OPEN];
   virtual void Guns(RTMessage)           = [ID_FIRST + ID_GUNS];
   virtual void DrawBorder(HDC hDC, int  x1, int y1, int x2, int y2,
                           BOOL isUp);
   virtual void UpdateList();
   virtual void UpdateButtonArrow();
   virtual void UpdateInfoArea();
   virtual void UpdateSheafWindow();
};


//====================================================================
// class                    SheafWndCls
//
// Purpose   : Creates a canvas for BurstsWndCls as a child window,
//             displays the bursts.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//====================================================================
class SheafWndCls : public TWindow
{
private:

   RECT    WndRect;
   HCURSOR hCursorTo;
   int     Cx;
   int     Cy;
   long    ScaleFactor;

public:

   SheafWndCls(PTWindowsObject AParent,
               LPSTR           ATitle,
               int X, int Y, int dX, int dY);
   ~SheafWndCls();
   virtual LPSTR GetClassName();
   virtual void  SetupWindow();
   virtual void  Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void  WMLButtonDown(TMessage &Msg)=[WM_FIRST +WM_LBUTTONDOWN];
   virtual void  WMSetCursor(TMessage &)     =[WM_FIRST + WM_SETCURSOR];
};
```

```
#endif __BURSTS_H
```

## Listing A.2: BUTTON2.H header file

```
//*********************************************************************
//
//                              button2.h
//
// header file for Button2Cls class
//*********************************************************************


#ifndef    __BUTTON2_H
#   define __BUTTON2_H

//note that the following code is executed only
//if this file has not been previously included



#ifdef __BORLANDC__
//  PC-specific includes
#   include <owl.h>
#   include <button.h>
#endif



#ifdef Unix
//   UNIX-specific includes
#endif



_CLASSDEF(Button2Cls)
//====================================================================
// class                         Button2Cls
//
// Purpose   : Creates a custom button. It inherits all the behavior
//             TButton class and adds the following features:
//                >It has a constant background.
//                >It may be instantiated with atmost two bitmap
//                 resource name. if it is intanctiated with two
//                 bitmaps, it displays the first one regularly and
//                 displays the second one when the button clicked.
//                 If it is instantiated with one bitmap only, then
//                 it displays that bitmap for all occassions.
//                 It may be instantiated with no bitmaps.
//                >Bitmap is displayed on the right of button while
//                 the text is on the left.
//
// Notes     : The size of bitmap file should be smaller than the
//             button sizes. Otherwise the bitmap is clipped.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
```

```
//===================================================================
class Button2Cls : public TButton
{
private:

   char Text[80];
   RECT BRect;
   HBITMAP hBitmap1;
   HBITMAP hBitmap2;

protected:

   virtual void ODADrawEntire(DRAWITEMSTRUCT _FAR & DrawInfo);
   virtual void ODAFocus(DRAWITEMSTRUCT _FAR & DrawInfo);
   virtual void ODASelect(DRAWITEMSTRUCT _FAR & DrawInfo);
   virtual void DrawFrame(HDC hDC, RECT Rect, BOOL Selected);
   virtual void WriteText(HDC hDC, LPSTR Text);
   virtual void PutBitmap(HDC hDC, BOOL Pressed = FALSE);

public:

   Button2Cls(PTWindowsObject AParent,
              int             AnId,
              LPSTR           AText,
              LPSTR           Bitmap1Name,
              LPSTR           Bitmap2Name,
              int             X,
              int             Y,
              int             W,
              int             H,
              BOOL            IsDefault,
              PTModule        AModule = NULL);
   ~Button2Cls();
};


#endif __BUTTON2_H
```

## Listing A.3: COMMAND.H header file

```
//*******************************************************************
//
//                             command.h
//
// header file for CommandWndCls,
//                 FireCommandsWndCls,
//                 InfoWndCls,
//                 SubFireCallWndCls, and
//                 ChronoWndCls classes
//*******************************************************************


#ifndef    __COMMAND_H
#   define __COMMAND_H

//note that the following code is executed only
```

```
//if this file has not been previously included


#include "fire.h"


#define MAXVLINE          6
#define MAXHLINE          8
#define ID_SEND         100
#define ID_FIRE         101
#define ID_INFO         201
#define ID_FIRECOMMANDS 301
#define ID_SUBFIRECALL  401

#define ID_CHRONO       501
#define ID_WHENREADY    502
#define ID_ATMYCOMMAND  503
#define ID_ATTIME       504
#define ID_AFTERTIME    505


_CLASSDEF(CommandWndCls)
_CLASSDEF(FireCommandsWndCls)
_CLASSDEF(InfoWndCls)
_CLASSDEF(SubFireCallWndCls)
_CLASSDEF(ChronoWndCls)
//======================================================================
// class                     CommandWndCls
//
// Purpose   : Hosts the other child windows.
//
// Notes     : It has one more constructure which is encapsulated as
//             protected. It is suppossed to be used only by its sub
//             classes.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//======================================================================
class CommandWndCls : public TWindow
{
private:

   RECT                  WndRect;
   PFireCommandsWndCls   FireCommandsWnd;
   PInfoWndCls           InfoWnd;
   PSubFireCallWndCls    SubFireCallWnd;
   PChronoWndCls         ChronoWnd;
   PButton2Cls           ButtonSend;
   PButton2Cls           ButtonFire;
   PButton2Cls           ButtonFireCommands;
   PButton2Cls           ButtonInfo;
   PButton2Cls           ButtonSubFireCall;
   PButton2Cls           ButtonChrono;

public:

   CommandWndCls(PTWindowsObject AParent,
                 LPSTR           ATitle,
                 int            X,
                 int            Y,
                 int            dX,
                 int            dY);
   ~CommandWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void Send(RTMessage)         = [ID_FIRST + ID_SEND];
   virtual void Fire(RTMessage)         = [ID_FIRST + ID_FIRE];
   virtual void FireCommands(RTMessage) = [ID_FIRST + ID_FIRECOMMANDS];
   virtual void Info(RTMessage)         = [ID_FIRST + ID_INFO];
   virtual void SubFireCall(RTMessage)  = [ID_FIRST + ID_SUBFIRECALL];
   virtual void ActivateChronometer(long dT);

protected:

   CommandWndCls(PTWindowsObject AParent,
                 LPSTR          ATitle);
};



//======================================================================
// class                    FireCommandsWndCls
//
// Purpose  : Display fire commands for each gun which is to fire.
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//======================================================================
class FireCommandsWndCls : public TWindow
{
private:

   RECT  WndRect;
   int   VLine[MAXVLINE-1];//x coordinates of vertical lines/borders
   int   HLine[MAXHLINE+1];//y coordiantes of horizantal lines/borders

public:

   FireCommandsWndCls(PTWindowsObject AParent,
                      LPSTR           ATitle,
                      int             X,
                      int             Y,
                      int             dX,
                      int             dY);
   ~FireCommandsWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
};


//======================================================================
// class                       InfoWndCls
//
// Purpose  : Displays fire call, fire order and message to observer
//            messages after they have been set.
```

```
//
// Notes      :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=============================================================
class InfoWndCls : public TWindow
{
private:

   RECT           WndRect;

public:

   InfoWndCls(PTWindowsObject AParent,
              LPSTR           ATitle,
              int X, int Y, int dX, int dY);
   ~InfoWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
};



//=============================================================
// class                       SubFireCallWndCls
//
// Purpose   : Displays subsequent fire calls.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=============================================================
class SubFireCallWndCls : public TWindow
{
private:

   RECT           WndRect;

public:

   SubFireCallWndCls(PTWindowsObject AParent,
                     LPSTR           ATitle,
                     int X, int Y, int dX, int dY);
   ~SubFireCallWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
};



//=============================================================
// class                       ChronoWndCls
//
// Purpose   : Displays a digital chronometer for seting the firing
//             time. It also instantiates the firing if it is a timed
//             firing.
//
// Notes     : Windows 3.1 allows only 16 clocks to work simultaneously,
//             each OFIRE program executes one already.
```

```
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//======================================================================
class ChronoWndCls : public TWindow
{
private:

   PTBRadioButton RadioWhenReady;
   PTBRadioButton RadioAtMyCommand;
   PTBRadioButton RadioAtTime;
   PTBRadioButton RadioAfterTime;
   RECT           WndRect;
   WORD           wTimer;
   WORD           wElapse;
   long           dTime;
   int            Hour;
   int            Minute;
   int            Second;

public:

   ChronoWndCls(PTWindowsObject AParent,
                LPSTR           ATitle,
                int             X,
                int             Y,
                int             dX,
                int             dY);
   ~ChronoWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void WMTimer(RTMessage)      = [WM_FIRST + WM_TIMER];
   virtual void WhenReady(RTMessage)    = [ID_FIRST + ID_WHENREADY];
   virtual void AtMyCommand(RTMessage) = [ID_FIRST + ID_ATMYCOMMAND];
   virtual void AtTime(RTMessage)       = [ID_FIRST + ID_ATTIME];
   virtual void AfterTime(RTMessage)    = [ID_FIRST + ID_AFTERTIME];
   virtual void StartCountDown(long dT);
};


#endif __COMMAND_H
```

## Listing A.4: FIRE.H header file

```
//**********************************************************************
//
//                                fire.h
//
// header file for dialog class declerations which are:
//          FireCallDlgCls,
//          PolarWndCls,
//          ShiftWndCls,
//          GridWndCls,
//          SuppressionWndCls,
//          FireOrderDlgCls,
//          MTODlgCls,
//          SubFireCallDlgCls,
//          TOTAtTimeDlgCls,
//          TOTAfterTimeDlgCls.
//*********************************************************************


#ifndef    __FIRE__H
#   define __FIRE__H


#ifdef __BORLANDC__
//  PC-specific includes
#   include <owl.h>
#   include <edit.h>
#   include <listbox.h>
#   include <combobox.h>
#   include <scrollba.h>
#   include <dialog.h>
#   include <bwcc.h>
#   include <button.h>
#   include <string.h>
#   include <bchkbox.h>
#   include <bradio.h>
#endif


#ifdef Unix
//   UNIX-specific includes
#endif


#include "button2.h"
#include "dbtarget.h"
#include "dbgun.h"


//firecall dialog
#define ID_FIREORDER        106
#define ID_VIEW             105
#define ID_FROM             107
#define ID_MISSIONTYPE      108
#define ID_SIZE             109
#define ID_POLAR            110
#define ID_SHIFT            111
#define ID_GRID             112
#define ID_SUPPRESSION      113
#define ID_DESCRIPTION      129

#define ID_TARGETNO         111

#define ID_DIRECTION        101
#define ID_DISTANCE         102
#define ID_UPDOWN           103
#define ID_SHIFTFROM        104
#define ID_RIGHTLEFT        105
#define ID_ADDDROP          106
#define ID_ZONE             107
#define ID_EAST             108
```

```
#define ID_NORTH            109
#define ID_HEIGHT           110

//mto dialog
#define ID_MTO              101
#define ID_PROCESS          110
#define ID_UNITTOFIRE       107
#define ID_GUNLIST          119
#define ID_ADJUSTINGELEMENT 108

//sub-fire call dialog
#define ID_DEVIATION        101
#define ID_DISTANCE         102
#define ID_HOB              103

#define ID_HOUR             701
#define ID_MINUTE           702
#define ID_SECOND           703


_CLASSDEF(PolarWndCls)
_CLASSDEF(ShiftWndCls)
_CLASSDEF(GridWndCls)
_CLASSDEF(SuppressionWndCls)
//====================================================================
// class                      FireCallDlgCls
//
// Purpose   : Demonstrates FireCall dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//====================================================================
class FireCallDlgCls : public TDialog
{
private:

   RECT cWnd;
   PPolarWndCls       PolarWnd;
   PShiftWndCls       ShiftWnd;
   PGridWndCls        GridWnd;
   PSuppressionWndCls SuppressionWnd;
   PTComboBox         ComboFrom;      //source of fire call
   PTComboBox         ComboType;      //type of mission
   PTComboBox         ComboSize;      //size of element for effect

public:
   FireCallDlgCls(PTWindowsObject AParent,
                  LPSTR           AName);
   ~FireCallDlgCls();
   virtual void SetupWindow();
   virtual void RadioPolar(RTMessage)      =[ID_FIRST + ID_POLAR];
   virtual void RadioShift(RTMessage)      =[ID_FIRST + ID_SHIFT];
   virtual void RadioGrid(RTMessage)       =[ID_FIRST + ID_GRID];
   virtual void RadioSuppression(RTMessage)=[ID_FIRST + ID_SUPPRESSION];
   virtual void ButtonOrder(RTMessage)      = [ID_FIRST + ID_FIREORDER];
   virtual void ButtonView(RTMessage)       = [ID_FIRST + ID_VIEW];
   virtual void UpdateFrom();
   virtual void UpdateSize();
   virtual void UpdateType();
   virtual void UpdateTargetDatabase();
};



//===================================================================
// class                        PolarWndCls
//
// Purpose   : Child window for FireCallDlgCls.
//             Displays polar location type controls.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//===================================================================
class PolarWndCls : public TWindow
{
private:

   RECT   WndRect;
   PTEdit EditDirection;
   PTEdit EditDistance;
   PTEdit EditUpDown;

public:

   PolarWndCls(PTWindowsObject AParent,
               LPSTR           ATitle,
               int, int, int, int);
   ~PolarWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC hDC, PAINTSTRUCT &);
   virtual void SetLocValues();
};



//===================================================================
// class                         ShiftWndCls
//
// Purpose   : Child window for FireCallDlgCls.
//             Displays shift location type controls.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//===================================================================
class ShiftWndCls : public TWindow
{
private:

   RECT          WndRect;
   PTComboBox    ComboFrom;
   PTEdit        EditDirection;
   PTEdit        EditRightLeft;
   PTEdit        EditAddDrop;
   PTEdit       EditUpDown;
   PDBTargetCls DBTargets;

public:

   ShiftWndCls(PTWindowsObject AParent,
               LPSTR           ATitle,
               int, int, int, int);
   ~ShiftWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC hDC, PAINTSTRUCT &);
   virtual void TargetNo(RTMessage) = [ID_FIRST + ID_SHIFTFROM];
   virtual void SetLocValues();
   virtual BOOL UpdateComboFrom();
};


//=================================================================
// class                         GridWndCls
//
// Purpose   : Child window for FireCallDlgCls.
//             Displays grid location type controls.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class GridWndCls : public TWindow
{
private:

   RECT   WndRect;
   PTEdit EditZone;
   PTEdit EditEast;
   PTEdit EditNorth;
   PTEdit EditAltitude;
   PTEdit EditDirection;

public:

   GridWndCls(PTWindowsObject AParent,
              LPSTR           ATitle,
              int, int, int, int);
   ~GridWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC hDC, PAINTSTRUCT &);
   virtual void SetLocValues();
};


//=================================================================
// class                       SuppressionWndCls
//
// Purpose   : Child window for FireCallDlgCls.
//             Displays suppression location type controls.
//
// Notes     :
//
```

```
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class SuppressionWndCls : public TWindow
{
private:

   RECT           WndRect;
   char           Zone[5];
   long           East;
   long           North;
   long           Height;
   PTComboBox     ComboTargetNo;
   PTEdit         EditDirection;
   PDBTargetCls   DBTargets;
   char           Description[80];

public:

   SuppressionWndCls(PTWindowsObject AParent,
                     LPSTR           ATitle,
                     int, int, int, int);
   ~SuppressionWndCls();

   virtual void  SetupWindow();
   virtual void  Paint(HDC hDC, PAINTSTRUCT &);
   virtual void  TargetNo(RTMessage) = [ID_FIRST + ID_TARGETNO];
   virtual void  SetLocValues();
   virtual BOOL  UpdateComboTargetNo();
   virtual char *GetTargetDescription();
};


//=====================================================================
// class                        FireOrderDlgCls
//
// Purpose   : Demonstrates FireOrder dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class FireOrderDlgCls : public TDialog
{
private:

   int           MainGun;
   int           GunNo[8];
   char          Zone[8][3];
   long          East[8];
   long          North[8];
   long          Height[8];
   PTComboBox    ComboUnitToFire;
   PTListBox     ListGuns;
   PTButton      ButtonAdjustingElement;
   PDBGunCls     DBGuns;

public:
   FireOrderDlgCls(PTWindowsObject AParent,
                   LPSTR           AName);
   ~FireOrderDlgCls();
   virtual void SetupWindow();
   virtual void MTO(RTMessage)              =[ID_FIRST + ID_MTO];
   virtual void ProcessForFire(RTMessage)   =[ID_FIRST + ID_PROCESS];
   virtual void UnitToFire(RTMessage)       =[ID_FIRST + ID_UNITTOFIRE];
   virtual void AdjustingElement(RTMessage)=[ID_FIRST +
                                             ID_ADJUSTINGELEMENT];
   virtual BOOL UpdateComboUnitToFire();
   virtual BOOL UpdateListGuns();
   virtual void CollectData();
};


//==================================================================
// class                          MTODlgCls
//
// Purpose   : Demonstrates message to observer dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//==================================================================
class MTODlgCls : public TDialog
{
public:

   MTODlgCls(PTWindowsObject AParent,
             LPSTR           AName);
   ~MTODlgCls();
   virtual void SetupWindow()
   {}
   virtual void Ok(RTMessage)        = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage)    = [ID_FIRST + IDCANCEL];
};


//==================================================================
// class                          SubFireCallDlgCls
//
// Purpose   : Demonstrates subfire call dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//==================================================================
class SubFireCallDlgCls : public TDialog
{
private:

   PTEdit   EditDeviation;
   PTEdit   EditDistance;
   PTEdit   EditHOB;
   PTButton ButtonProcess;

public:

   SubFireCallDlgCls(PTWindowsObject AParent,
                     LPSTR           AName);
   ~SubFireCallDlgCls();
   virtual void Process(RTMessage)  = [ID_FIRST + ID_PROCESS];
   virtual void Cancel(RTMessage)   = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)     = [ID_FIRST + IDHELP];
};


//=====================================================================
// class                          TOTAtTimeDlgCls
//
// Purpose   : Demonstrates (time on target) At Time dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//=====================================================================
class TOTAtTimeDlgCls : public TDialog
{
private:

   PTEdit   EditHour;
   PTEdit   EditMinute;
   PTEdit   EditSecond;

public:

   TOTAtTimeDlgCls(PTWindowsObject AParent,
                   LPSTR           AName);
   ~TOTAtTimeDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage)       = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage)   = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)     = [ID_FIRST + IDHELP];
};


//=====================================================================
// class                          TOTAfterTimeDlgCls
//
// Purpose   : Demonstrates (time on target) After Time dialog box
//
// Notes     : It work with Paradox Engine, make sure related *.dll
//             files are on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//=====================================================================
class TOTAfterTimeDlgCls : public TDialog
{
private:
   PTEdit   EditHour;
   PTEdit   EditMinute;
   PTEdit   EditSecond;

public:

   TOTAfterTimeDlgCls(PTWindowsObject AParent, LPSTR AName);
   ~TOTAfterTimeDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage)     = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage) = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)   = [ID_FIRST + IDHELP];
};


#endif __FIRE__H
```

## Listing A.5: GLOBALS.H header file

```
//*********************************************************************
//
//                                globals.h
//
// header file for global variables, objects and
//          ComputationCls
//*********************************************************************



#ifndef    __GLOBALS__H
#   define __GLOBALS__H



#include <string.h>
#include "dbtbl_f.h"



#define SUCCESS 1
#define FAIL    0



enum MissionType   {mADJUSTFIRE,
                    mFIREFOREFFECT,
                    mSUPPRESSION,
                    mIMMEDIATESUPPRESSION,
                    mIMMEDIATESMOKE};

enum LocationType {POLAR,
                   SHIFT,
                   GRID,
                   SUPPRESSION};

enum SheafType    {CONVERGED,
                   PARALLEL,
                    SPECIAL,
                    OPEN);

enum TimingType     {WHENREADY,
                     ATMYCOMMAND,
                     ATTIME,
                     AFTERTIME};

//===================================================================
//struct                          Gun
//
// Purpose  : Holds information for a gun
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
struct Gun
{
   int    GunNo;

   //gun location
   char   Zone[5];
   double East;
   double North;
   double Height;

   //firing command elements
   double Timing;
   double Deflection;
   double Elevation;
   double Quadrant;

   //target-hit info
   char   HitZone[5];
   double HitEast;
   double HitNorth;
   double HitHeight;
   double HitRange;
   double HitDeflection; //chart deflection
   double HitAzimuth;    //gun-hit point azimuth

   //gun general info
   double CommonOrientation;
};


//_CLASSDEF(ComputationCls)
//===================================================================
// class                    ComputationCls
//
// Purpose  : Makes all the computations for a gun battery fire direction
//
// Notes    : It works with Paradox Engine, so necessary *.dll files
//            should be on path.
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class ComputationCls
{
private:

   WStr           FiringTableName;
   WStr           FiringCharge;
   PDBTable_F_Cls DBTable_F;

public:

   char           BatteryName[3];
   char           CallLine[4][80];
   char           OrderLine[80];
   char           MTOLine[80];
   char           SubCallLine[50];

   //observer information
   char   ObserverName[10];
   char   ObserverZone[10];
   double ObserverEast;
   double ObserverNorth;
   double ObserverHeight;
   double OTDirection;

   //target location before adjustment
   char   IniTargetZone[3];
   double IniTargetEast;
   double IniTargetNorth;
   double IniTargetHeight;
   double IniTargetRange;     //for main gun
   double IniTargetDeflection;//for main gun
   //target location after adjustment
   char   CurrTargetZone[3];
   double CurrTargetEast;
   double CurrTargetNorth;
   double CurrTargetHeight;

   //target location settings
   //
   //polar
   double PolarDistance;
   double PolarUpDown;
   //
   //grid
   char   GridZone[3];
   double GridEast;
   double GridNorth;
   double GridHeight;
   //
   //shift
   chaı   ShiftFrom[10];
   char   ShiftZone[10];
   double ShiftEast;
   double ShiftNorth;
   double ShiftHeight;
   double ShiftRightLeft;
   double ShiftAddDrop;
   double ShiftUpDown;
   //
   //suppression
   char   SuppTargetNo[10];
   char   Description[128];

   char   Category[80];

   //fire command settings
   int NoOfPiecesToFollow;

   //current values
   char          CurrFrom[10];
   char          CurrMissionType[50];
   char          CurrSize[10];
   LocationType  CurrLocationType;
   SheafType     CurrSheaf;

   //gun parameters
   Gun    Guns[8];
   int    MainGun;
   double AzimuthOfFire;

   //gun - ammo
   double BurstDiameter;

   //fire commands buffers
   int FiredNo;

   ComputationCls();
   ~ComputationCls();

   virtual void ComputeHitCoors();
   virtual void ComputeChartValues();
   virtual void ComputeRangeAzimuth(double  x1,
                                    double  y1,
                                    double  x2,
                                    double  y2,
                                    double &Range,
                                    double &Azimuth);
   virtual void ComputeCoordinate(double  x1,
                                  double  y1,
                                  double  Range,
                                  double  Azimuth,
                                  double &x2,
                                  double &y2);
   void SetFiringTableName(WStr FTName);
   WStr GetFiringCharge();
   BOOL ComputeFiringCharge();
   BOOL ComputeElevations();
   BOOL ComputeFiringValues();

   void Reset();
};

#endif __GLOBALS__H
```

## Listing A.6: REPORT.H header file

```
//**********************************************************************
//
```

```
//                              report.h
//
// header file for the classes ReportWndCls,
//                              TextWndCls
//*******************************************************************


#ifndef    __REPORT_H
#   define __REPORT_H

// Note the following code is only executed if
// this file has not been previously included


#include "fire.h"


#define ID_PRINT 101


_CLASSDEF(ReportWndCls)
_CLASSDEF(TextWndCls)
//===================================================================
// class                      ReportWndCls
//
// Purpose   : Hosts a canvas window which displays the firig report
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class ReportWndCls : public TWindow
{
private:

   RECT        WndRect;
   PTextWndCls TextWnd;
   PButton2Cls ButtonPrint;

public:

   ReportWndCls(PTWindowsObject AParent,
                LPSTR           ATitle,
                int             X,
                int             Y,
                int             dX,
                int             dY);
   ~ReportWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void Print(RTMessage)  = [ID_FIRST + ID_PRINT];
};
```

```
//=================================================================
// class                        TextWndCls
//
// Purpose   : Serves as a canvas to display the firing report
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class TextWndCls : public TWindow
{
private:

   RECT        WndRect;

public:
   TextWndCls(PTWindowsObject AParent,
              LPSTR           ATitle,
              int             X,
              int             Y,
              int             dX,
              int             dY);
   ~TextWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
};


#endif __REPORT_H
```

## Listing A.7: SELECT.H header file

```
//*******************************************************************
//
//                                  select.h
//
// header file for the class SelectionWndCls
//*******************************************************************


#ifndef     __SELECT_H
#   define  __SELECT_H

// Note the following code is only executed if
// this file has not been previously included


#include "fire.h"
#include "command.h"
#include "bursts.h"
#include "report.h"
```

```
#define ID_EXIT 101
#define ID_HELP 102


//from fire.cpp
extern PCommandWndCls CommandWnd;
extern PBurstsWndCls  BurstsWnd;
extern PReportWndCls  ReportWnd;


_CLASSDEF(SelectionWndCls)
//====================================================================
// class                      SelectionWndCls
//
// Purpose   : Displays notebook tab-stops to select different windows.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//====================================================================
class SelectionWndCls : public TWindow
{
private:

   PTWindowsObject MainWnd;
   RECT            WndRect;
   POINT           IndexTab[3][4];
   short           CurrIndexTab;
   PButton2Cls     ButtonExit;
   PButton2Cls     ButtonHelp;

public:

   SelectionWndCls(PTWindowsObject AParent,
                   LPSTR           ATitle,
                   int             X,
                   int             Y,
                   int             dX,
                   int             dY);
   ~SelectionWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void WMLButtonDown(TMessage &) = [WM_FIRST + WM_LBUTTONDOWN];
   virtual void Exit(RTMessage)           = [ID_FIRST + ID_EXIT];
   virtual void Help(RTMessage)           = [ID_FIRST + ID_HELP];
};


#endif
```

# APPENDIX B. (OCHART HEADER FILES LISTINGS)

## Listing B.1: ABSLIST.H header file

```
//**********************************************************************
//
//                                  abslist.h
//
// header file for the class genAbstractList
//**********************************************************************



#ifndef    _ABSLIST_H
#   define _ABSLIST_H

//note that the following code is executed only
//if this file has not been previously included



#define ALLOCATE_ERROR "Dynamic allocation error"



enum boolean { false, true };

typedef char string80[81];



//=====================================================================
// template class             genAbstractList
//
// Purpose   : implements an abstract class of linked lists with
//             the following operations and features:
//
//             > nodes with duplicate keys can be allowed.
//             > insert new node.
//             > search for a node with a specific occurrence
//               of a key.
//             > delete a node with a specific occurrence of a key.
//             > traverse the nodes of the lists.
//               the user to modify them.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//=====================================================================
template<class T>
class genAbstractList
{
protected:

   unsigned listSize;       // number of nodes
   boolean  overwrite;      // overwrite data flag
   boolean  hasDuplicate;   // duplicate-key flag
   string80 errorMessage;   // error message
```

```
public:

   //state query methods
   boolean isEmpty()
   {
      return (listSize == 0) ? true : false;
   }


   unsigned getListSize()
   {
      return listSize;
   }


   char* getErrorMessage()
   {
      string80 s;
      strcpy(s, errorMessage);
      errorMessage[0] = '\0';
      return s;
   }


   //object manipulation methods
   genAbstractList()
   {}


   virtual boolean insert(T&)
   {
      return false;
   }


   virtual boolean remove(T&, unsigned occur = 1)
   {
      return false;
   }


   virtual boolean search(T&, unsigned occur = 1)
   {
       return false;
   }


   virtual boolean visitFirstNode(T&)
   {
       return false;
   }
   virtual boolean visitNextNode(T&)
   {
       return false;
   }


   virtual void clear()
   {}
};


#endif _ABSLIST_H
```

## Listing B.2: CHART.H header file

```
//*********************************************************************
//
//                              chart.h
//
// header file the classes GridSetupDlgCls
//                         TargetSetupDlgCls
//                         GunSetupDlgCls
//                         NewEntryDlgCls
//                         NewFromEntryDlgCls
//                         GunInputDlgCls
//                         TargetInputDlgCls
//                         TargetDBDlgCls
//                         GunDBDlgCls
//*********************************************************************


#ifndef    __CHART_H
#   define __CHART_H

//note that the following code is executed only
//if this file has not been previously included


#include <owl.h>
#include <edit.h>
#include <listbox.h>
#include <combobox.h>
#include <dialog.h>
#include <bwcc.h>
#include <button.h>
#include <string.h>
#include <static.h>
#include <scrollba.h>
#include <bchkbox.h>
#include <bradio.h>

#include "defines.h"
#include "globals.h"
```

```
#include "targetdb.h"
#include "gundb.h"


//===================================================================
// class                    GridSetupDlgCls
//
// Purpose   : Displays grid setup dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class GridSetupDlgCls : public TDialog
{
private:

   PTEdit     EditZone;
   PTEdit     EditLeft;
   PTEdit     EditRight;
   PTEdit     EditBottom;
   PTEdit     EditTop;
   PTComboBox ComboDistance;
   PTComboBox ComboScale;
   PTCheckBox CheckShowGrid;
   PTCheckBox CheckShowNorth;

public:

   GridSetupDlgCls(PTWindowsObject AParent, LPSTR AName);
   ~GridSetupDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage)          = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage)      = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)        = [ID_FIRST + IDHELP];
};


//===================================================================
// class                    TargetSetupDlgCls
//
// Purpose   : Displays target setup dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class TargetSetupDlgCls : public TDialog
{
private:

   PTCheckBox CheckShowTargets;

public:

   TargetSetupDlgCls(PTWindowsObject AParent, LPSTR AName);
   ~TargetSetupDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage)              = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage)          = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)            = [ID_FIRST + IDHELP];

};


//=====================================================================
// class                         GunSetupDlgCls
//
// Purpose   : Displays gun setup dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class GunSetupDlgCls : public TDialog
{
private:

   PTComboBox ComboScale;
   PTCheckBox CheckShowCoverings;

public:

   GunSetupDlgCls(PTWindowsObject AParent, LPSTR AName);
   ~GunSetupDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage)              = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage)          = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)            = [ID_FIRST + IDHELP];
};


//=====================================================================
// class                         NewEntryDlgCls
//
// Purpose   : Displays new entry dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class NewEntryDlgCls : public TDialog
{
protected:

   long           NewX;
   long           NewY;
   PTRadioButton RadioTarget;
   PTRadioButton RadioGun;
   PTRadioButton RadioRadar;
   PTRadioButton RadioObservationPost;
   PTRadioButton RadioRegPoint;
   PTRadioButton RadioCheckMark;
```

```
public:

   NewEntryDlgCls(PTWindowsObject AParent,
                  LPSTR           AName,
                  long            X,
                  long            Y);
   ~NewEntryDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage Msg)     = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage Msg) = [ID_FIRST + IDCANCEL];
};


//=================================================================
// class                      NewFromEntryDlgCls
//
// Purpose  : Displays new from entry dialog box.
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class NewFromEntryDlgCls : public NewEntryDlgCls
{
private:

   long    FromX;
   long    FromY;
   int     OffAzimuth;
   long    OffDistance;
   PTEdit  EditEast;
   PTEdit  EditNorth;
   PTEdit  EditAzimuth;
   PTEdit  EditDistance;

public:

   NewFromEntryDlgCls(PTWindowsObject AParent,
                      LPSTR           AName,
                      long            X,
                      long            Y,
                      int             Azimuth,
                      long            Distance);
   ~NewFromEntryDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage Msg)     = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage Msg) = [ID_FIRST + IDCANCEL];
};


//=================================================================
// class                      GunInputDlgCls
//
// Purpose  : Displays gun input dialog box.
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//==============================================================
class GunInputDlgCls : public TDialog
{
private:

   GunDBCls     aGun;
   PTEdit       EditEast;
   PTEdit       EditNorth;
   PTEdit       EditHeight;
   PTEdit       EditRange;
   PTEdit       EditLeft;
   PTEdit       EditRight;
   PTEdit       EditCommonDef;
   PTEdit       EditOrientation;
   PTComboBox   ComboBattalion;
   PTComboBox   ComboBattery;
   PTComboBox   ComboGun;
   PTComboBox   ComboGunType;
   PTBCheckBox  CheckMainGun;

public:

   GunInputDlgCls(PTWindowsObject AParent,
                  LPSTR           AName,
                  long            X,
                  long            Y);
   GunInputDlgCls(PTWindowsObject AParent,
                  LPSTR           AName,
                  GunDBCls        Gun);
   ~GunInputDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage Msg)     = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage Msg) = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage Msg)   = [ID_FIRST + IDHELP];
};


//==============================================================
// class                     TargetInputDlgCls
//
// Purpose   : Displays target input dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//==============================================================
class TargetInputDlgCls : public TDialog
{
private:

   TargetDBCls   aTarget;
   PTEdit        EditTargetNo;
   PTEdit        EditEast;
   PTEdit        EditNorth;
   PTEdit        EditHeight;
   PTEdit        EditDescription;
   PTComboBox    ComboCategory;
public:

   TargetInputDlgCls(PTWindowsObject AParent,
                     LPSTR           AName,
                     long            X,
                     long            Y);
   TargetInputDlgCls(PTWindowsObject AParent,
                     LPSTR           AName,
                     TargetDBCls     Tar);
   ~TargetInputDlgCls();
   virtual void SetupWindow();
   virtual void Ok(RTMessage Msg)     = [ID_FIRST + IDOK];
   virtual void Cancel(RTMessage Msg) = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage Msg)   = [ID_FIRST + IDHELP];
};


//===================================================================
// class                      TargetDBDlgCls
//
// Purpose   : Displays target database dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class TargetDBDlgCls : public TDialog
{
private:

   PTListBox       ListNames;
   PTWindowsObject MainWnd;

public:

   TargetDBDlgCls(PTWindowsObject AParent,
                  LPSTR           AName);
   ~TargetDBDlgCls();
   virtual void SetupWindow();
   virtual void UpdateListBox();
   virtual void ButtonAdd(RTMessage)    = [ID_FIRST + ID_ADD];
   virtual void ButtonDelete(RTMessage) = [ID_FIRST + ID_DELETE];
   virtual void ButtonDetail(RTMessage) = [ID_FIRST + ID_DETAIL];
   virtual void ButtonShow(RTMessage)   = [ID_FIRST + ID_SHOW];
   virtual void Cancel(RTMessage)       = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)         = [ID_FIRST + IDHELP];
};


//===================================================================
// class                      GunDBDlgCls
//
// Purpose   : Displays gun database dialog box.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
```

```
//======================================================================
class GunDBDlgCls : public TDialog
{
private:

   PTListBox        ListNames;
   PTWindowsObject MainWnd;

public:

   GunDBDlgCls(PTWindowsObject AParent,
               LPSTR           AName);
   ~GunDBDlgCls();
   virtual void SetupWindow();
   virtual void UpdateListBox();
   virtual void ButtonAdd(RTMessage)    = [ID_FIRST + ID_ADD];
   virtual void ButtonDelete(RTMessage) = [ID_FIRST + ID_DELETE];
   virtual void ButtonDetail(RTMessage) = [ID_FIRST + ID_DETAIL];
   virtual void ButtonShow(RTMessage)   = [ID_FIRST + ID_SHOW];
   virtual void Cancel(RTMessage)       = [ID_FIRST + IDCANCEL];
   virtual void Help(RTMessage)         = [ID_FIRST + IDHELP];
};



#endif __CHART_H
```

## Listing B.3: DEFINES.H header file

```
//**********************************************************************
//
//                              defines.h
//
// header file for defines
//**********************************************************************



#ifndef    __DEFINES_H
#   define __DEFINES_H


//menu identifiers
#define CM_TRAININGMODE    101
#define CM_EXITCHART       109
#define CM_INSERT          203
#define CM_INSERTFROM      204
#define CM_GRIDSETUP       303
#define CM_TARGETSETUP     304
#define CM_GUNSETUP        305
#define CM_SAVEONEXIT      309
#define CM_DATABASETARGET  401
#define CM_DATABASEGUN     402
#define CM_ABOUT           151
```

```
//info window menu identifiers
#define CM_SCALEONE          801
#define CM_SCALETWO          802
#define CM_SCALETHREE        803
#define CM_SCALEFOUR         804
#define CM_SCALEFIVE         805
#define CM_SCALESIX          806

//Grid setup dialog window
#define ID_ZONE              112
#define ID_LEFT              105
#define ID_BOTTOM            106
#define ID_RIGHT             107
#define ID_TOP               108
#define ID_DISTANCE          204
#define ID_SCALE             110
#define ID_SHOWGRIDLINES     101
#define ID_SHOWNORTHSIGN     102

//TargetSetupDlgCls dialog window
#define ID_SHOWTARGETS       110

//GunSetupDlgCls dialog window
#define ID_SHOWGUNCOVERINGS 101

//NewEntryDlgCls dialog window
#define ID_TARGET            101
#define ID_GUN               102
#define ID_RADAR             103
#define ID_OBSERVATIONPOST   104
#define ID_REGPOINT          105
#define ID_CHECKMARK         106

//NewFromEntryDlgCls dialog window
//it uses id_east, id_north, id_target, id_gun, id_others as well
#define ID_EAST              201
#define ID_NORTH             202
#define ID_AZIMUTH           203
#define ID_OFFDISTANCE       111

//GunInputDlgCls dialog window
#define ID_MAINGUN           999
#define ID_BATTERYNAME      1000
#define ID_GUNNO            1001
#define ID_BATTALIONNAME    1002
#define ID_GUNTYPE          1005
#define ID_HEIGHT           1009
#define ID_MAXRANGE         1010
#define ID_MAXLEFT          1011
#define ID_MAXRIGHT         1012
#define ID_COMMONDEF        1013
#define ID_ORIENTATION      1014

//TargetInputDlgCls dialog window
#define ID_TARGETNO          103
#define ID_DESCRIPTION       107
#define ID_CATEGORY          108

//TargetDBDlgCls dialog window
#define ID_NAMES             101
#define ID_ADD               102
```

```
#define ID_DELETE          103
#define ID_DETAIL          104
#define ID_SHOW            105


#endif __DEFINES_H
```

## Listing B.4: GEN2LIST.H header file

```
//*********************************************************************
//
//                            gen2list.h
//
//  header file for the template classes
//           genOdList
//           genUdList
//*********************************************************************


#ifndef    _GEN2LIST_H
#   define _GEN2LIST_H

//note that the following code is executed only
//if this file has not been previously included


#include "abslist.h"


//=====================================================================
// template struct              dListNode
//=====================================================================
template<class T>
struct dListNode {
   T dataNode;
   dListNode<T> *prevPtr;
   dListNode<T> *nextPtr;
};


//=====================================================================
// template class              genOd.ist
//
// Purpose   : implements a classes of ordered doubly-linked lists
//             with the following operations and features:
//
//              > nodes with duplicate keys can be allowed.
//              > insert new node.
//              > search for a node with a specific occurrence
//                of a key.
//              > delete a node with a specific occurrence of a key.
//              > traverse the nodes of the lists.
//
```

```
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
template<class T>
class genOdList : public genAbstractList<T>
{
protected:
   dListNode<T> *nodePtr, // node-visitation pointer
                *tail,    // pointer to the tail of list
                *head;    // pointer to head of list
   T            _buf1;    // buffer

   genOdList<T>& copy(genOdList<T>&);

public:

   //state manipulation methods
   genOdList()
   { }



   genOdList(boolean canoverwrite, boolean allowduplicate);



   genOdList(genOdList<T>& dl)
   {
      head = NULL; copy(dl);
   }



   ~genOdList()
   {
      clear();
   }



   void setNodePtr(dListNode<T>* listPtr);



   void setBuf1(T& newBuf1)
   {
      _buf1 = newBuf1;
   }


   //state query methods
   dListNode<T>* getNodePtr()
   {
      return nodePtr;
   }



   //object manipulation methods
   virtual boolean insert(T&);
   virtual boolean remove(T& x, unsigned occur = 1);


   virtual boolean search(T& x, unsigned occur = 1)
   {
      dListNode<T> *p;
    return search(p, x, occur);
   }


   virtual boolean search(dListNode<T>* &thisptr,
                          T&            x,
                          unsigned      occur = 1);

   virtual boolean visitFirstNode(T& x);
   boolean visitPrevNode(T& x);
   virtual boolean visitNextNode(T& x);
   boolean visitLastNode(T& x);
   virtual void clear();


   genOdList<T>& operator =(genOdList<T>& dl)
   {
      copy(dl);
      return *this;
   }
};


//=====================================================================
// template class                genUdList
//
// Purpose  : implements a classes of unordered doubly-linked lists
//            with the following operations and features:
//
//             > nodes with duplicate keys can be allowed.
//             > insert new node.
//             > search for a node with a specific occurrence
//               of a key.
//             > delete a node with a specific occurrence of a key.
//             > traverse the nodes of the lists.
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
template<class T>
class genUdList : public genOdList<T>
{
public:

   //state manipulation methods
   genUdList() {}

   genUdList(boolean canoverwrite, boolean allowduplicate)
   :genOdList<T>(canoverwrite, allowduplicate)
   { }


   genUdList(genUdList<T>& dl)
   {
      head = NULL; copy(dl);
   }


   //object manipulation methods
   virtual boolean insert(T&);
   virtual boolean search(dListNode<T>* &thisptr,
                          T&               x,
                          unsigned         occur = 1);


   genUdList<T>& operator = (genUdList<T>& dl)
   {
      copy(dl);
      return *this;
   }
};


#endif _GEN2LIST_H
```

## Listing B.5: GLOBALS.H header file

```
//*********************************************************************
//
//                              globals.h
//
// header file for the class GlobalCls and varibals
//*********************************************************************



#ifndef    __GLOBALS_H
#   define __GLOBALS_H

//note that the following code is executed only
//if this file has not been previously included


#define FACTORSLIDE        250//the bigger the slower


//====================================================================
// class                         GlobalsCls
//
// Purpose  : Holds the global objects and variables for the rest of
//            the application.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
class GlobalsCls
{
public:

   long  WCLeft, WCBottom;  //lower left most coors of world
   long  VisibleWCLeft;     //left-most coordinate of screen
   long  VisibleWCBottom;   //bottom-most coordinate of screen
   long  WCToRightExtension;//just for scroll bars limitations
   long  WCToTopExtension;  //just for scroll bar limitations
   int   Distances[5];
   int   CurrDistanceIndex;
   short DistancesCount;
   long  Scales[6];
   int   CurrScaleIndex;
   short ScalesCount;
   BOOL  ShowGrids;
   BOOL  ShowNorthArrow;
   BOOL  ShowTargets;
   BOOL  ShowGunCoverings;
   BOOL  SaveOnExit;
   int   MinScaleNDXForGuns;
   short CurrBmpButtonSelection; //tool bar bitmap menu selections
   long  SlideAmount;            //scroll bar sliding amount in meters

   //main window dimensions
   int   WidthMainWindow;
   int   HeightMainWindow;

   //...............................................................
   GlobalsCls()
   //...............................................................
   {
      Reset();
   }


   //...............................................................
   Reset()
   //...............................................................
   {
      //firing chart extremum coordinates
      WCLeft   = 10000;
      WCBottom = 10000;
      VisibleWCLeft      = WCLeft;
      VisibleWCBottom    = WCBottom;
      WCToRightExtension = 50000;
      WCToTopExtension   = 50000;

      Distances[0]      =  100;
      Distances[1]      =  500;
      Distances[2]      = 1000;
      Distances[3]      = 2000;
      Distances[4]      = 5000;
      CurrDistanceIndex =    2;
      DistancesCount      =     5;
      Scales[0]           =   2500;
      Scales[1]           =   5000;
      Scales[2]           =  12500;
      Scales[3]           =  25000;
      Scales[4]           =  50000;
      Scales[5]           = 100000;
      CurrScaleIndex      =      2;
      ScalesCount         =      6;

      ShowGrids           = TRUE;
      ShowTargets         = TRUE;
      ShowGunCoverings    = TRUE;
      MinScaleNDXForGuns  = 1;
      ShowNorthArrow      = TRUE;
      SaveOnExit          = TRUE;

      CurrBmpButtonSelection = -1; //means none of them is selected

      SlideAmount         = Scales[CurrScaleIndex] / FACTORSLIDE;
      WidthMainWindow     = 800;
      HeightMainWindow    = 600;
   }

};

#endif __GLOBALS_H
```

## Listing B.6: GUNDB.H header file

```
//*********************************************************************
//
//                              gundb.h
//
// header file for the class GunDBCls
//*********************************************************************


#ifndef    _GUNDB_H
#   define _GUNDB_H

//note that the following code is executed only
//if this file has not been previously included


#include <string.h>


#define MAXNAMELEN   10
#define MAXMODELLEN  30


_CLASSDEF(GunDBCls)
//=====================================================================
// class                          GunDBCls
```

```
//
// Purpose  :Serves as database management class for gun database
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class GunDBCls
{
private:

   char           BattalionName[MAXNAMELEN];
   char           BatteryName[MAXNAMELEN];
   short          GunNo;
   long           CoorEast;
   long           CoorNorth;
   int            Height;
   BOOL           MainGun;
   char           Model[MAXMODELLEN];
   long           MaxRange;
   int            MaxRightDeflection;
   int            MaxLeftDeflection;
   int            CommonOrientationDeflection;
   int            OrientationDeflection;

public:

   //.................................................................
   GunDBCls()
   //.................................................................
   {
      strcpy(BattalionName, "*");
      BatteryName[0]                 = NULL;
      GunNo                          = 0;
      CoorEast                       = 0;
      CoorNorth                      = 0;
      Height                         = 0;
      MainGun                        = 0;
      strcpy(Model, "[-]");
      MaxRange                       = 0;
      MaxRightDeflection             = 0;
      MaxLeftDeflection              = 0;
      CommonOrientationDeflection    = 0;
      OrientationDeflection          = 0;
   }


   //.................................................................
   GunDBCls(char* Btn,
            char* Btr,
            int   Gun,
            long  East,
            long  North,
            int   Height,
            BOOL  mGun,
            char* Mod,
            long  Range,
            int   DefR,
            int   DefL,
         int   CommonDef,
         int   Orientation)
//...............................................................
{
   strcpy(BattalionName, Btn);
   strcpy(BatteryName, Btr);
   GunNo                            = Gun;
   CoorEast                         = East;
   CoorNorth                        = North;
   Height                           = Height;
   MainGun                          = mGun;
   strcpy(Model, Mod);
   MaxRange                         = Range;
   MaxRightDeflection               = DefR;
   MaxLeftDeflection                = DefL;
   CommonOrientationDeflection      = CommonDef;
   OrientationDeflection            = Orientation;
}


//...............................................................
GunDBCls(const GunDBCls& T)
//...............................................................
//copy initializer
{
   strcpy(BattalionName, T.BattalionName);
   strcpy(BatteryName, T.BatteryName);
   GunNo                            = T.GunNo;
   CoorEast                         = T.CoorEast;
   CoorNorth                        = T.CoorNorth;
   Height                           = T.Height;
   MainGun                          = T.MainGun ;
   strcpy(Model, T.Model);
   MaxRange                         = T.MaxRange;
   MaxRightDeflection               = T.MaxRightDeflection;
   MaxLeftDeflection                = T.MaxLeftDeflection;
   CommonOrientationDeflection      = T.CommonOrientationDeflection;
   OrientationDeflection            = T.OrientationDeflection;
}

//...............................................................
~GunDBCls()
//...............................................................
{}


//assignment operator
//...............................................................
GunDBCls& operator = (const GunDBCls &T)
//...............................................................
{
   strcpy(BattalionName, T.BattalionName);
   strcpy(BatteryName, T.BatteryName);
   GunNo                            = T.GunNo;
   CoorEast                         = T.CoorEast;
   CoorNorth                        = T.CoorNorth;
   Height                           = T.Height;
   MainGun                          = T.MainGun ;
   strcpy(Model, T.Model);
   MaxRange                        = T.MaxRange;
   MaxRightDeflection              = T.MaxRightDeflection;
   MaxLeftDeflection               = T.MaxLeftDeflection;
   CommonOrientationDeflection     = T.CommonOrientationDeflection;
   OrientationDeflection           = T.OrientationDeflection;
   return *this;
 }


//...................................................................
int operator <  (GunDBCls& aGun)
//...................................................................
{
   if( strcmp(BattalionName, aGun.BattalionName) < 0 ) {
      return 1;
   }else if(strcmp(BattalionName, aGun.BattalionName) == 0 ) {
      if(strcmp(BatteryName, aGun.BatteryName) < 0 ) {
         return 1;
      }else if(strcmp(BatteryName, aGun.BatteryName) == 0 ) {
         if( GunNo < aGun.GunNo )
            return 1;
      }
   }
   return 0;
}


//...................................................................
int operator >  (GunDBCls& aGun)
//...................................................................
{
   if( strcmp(BattalionName, aGun.BattalionName) > 0 ) {
      return 1;
   }else if(strcmp(BattalionName, aGun.BattalionName) == 0 ) {
      if(strcmp(BatteryName, aGun.BatteryName) > 0 ) {
         return 1;
      }else if(strcmp(BatteryName, aGun.BatteryName) == 0 ) {
         if( GunNo > aGun.GunNo )
            return 1;
      }
   }
   return 0;
}


//...................................................................
int operator <=  (GunDBCls& aGun)
//...................................................................
{
   if( strcmp(BattalionName, aGun.BattalionName) <= 0 ) {
      return 1;
   }else if(strcmp(BattalionName, aGun.BattalionName) == 0 ) {
      if(strcmp(BatteryName, aGun.BatteryName) <= 0 ) {
         return 1;
      }else if(strcmp(BatteryName, aGun.BatteryName) == 0 ) {
         if( GunNo <= aGun.GunNo )
            return 1;
         }
      }
      return 0;
   }


   //.....................................................................
   int operator == (GunDBCls& aGun)
   //.....................................................................
   {
      if(!strcmp(BattalionName, aGun.BattalionName))
         if(!strcmp(BatteryName, aGun.BatteryName))
            if(GunNo == aGun.GunNo)
               return 1;
      return 0;
   }


   //---functions to access the data members
   char *GetBattalionName()      {return BattalionName;}
   char *GetBatteryName()        {return BatteryName;}
   short GetGunNo()              {return GunNo;}
   long  GetCoorEast()           {return CoorEast;}
   long  GetCoorNorth()          {return CoorNorth;}
   int   GetHeight()             {return Height;}
   BOOL  isMainGun()             {return MainGun;}
   char *GetModel()              {return Model;}
   long  GetMaxRange()           {return MaxRange;}
   int   GetMaxRightDeflection() {return MaxRightDeflection;}
   int   GetMaxLeftDeflection()  {return MaxLeftDeflection;}
   int   GetCommonOrientationDeflection() {return
CommonOrientationDeflection;}
   int   GetOrientationDeflection()       {return OrientationDeflection;}

   //---functions to set the data members
   void SetBattalionName(char *in)       {strcpy(BattalionName, in);}
   void SetBatteryName(char *in)         {strcpy(BatteryName, in);}
   void SetGunNo(short in)               {GunNo = in;}
   void SetCoorEast(long in)             {CoorEast = in;}
   void SetCoorNorth(long in)            {CoorNorth = in;}
   void SetHeight(int in)                {Height = in;}
   void SetMainGun(BOOL in)              {MainGun = in;}
   void SetModel(char *in)               {strcpy(Model, in);}
   void SetMaxRange(long in)             {MaxRange = in;}
   void SetMaxLeftDeflection(int in)     {MaxLeftDeflection = in;}
   void SetMaxRightDeflection(int in)    {MaxRightDeflection = in;}
   void SetCommonOrientationDeflection(int in)
{CommonOrientationDeflection = in;}
   void SetOrientationDeflection(int in)       {OrientationDeflection =
in;}



   //.....................................................................
   void Reset()
   //.....................................................................
   //resets all data members
   {
        BattalionName[0]              = NULL;
        BatteryName[0]                = NULL;
        GunNo                         = 0;
        CoorEast                      = 0;
        CoorNorth                     = 0;
        Height                        = 0;
        MainGun                       = 0;
        strcpy(Model, "[-]");
        MaxRange                      = 0;
        MaxRightDeflection            = 0;
        MaxLeftDeflection             = 0;
        CommonOrientationDeflection   = 0;
        OrientationDeflection         = 0;
    }

};

#endif _GUNDB_H
```

## Listing B.7: INFO.H header file

```
//*******************************************************************
//
//                                 info.h
//
// header file for the classes SelectionToolCls
//                             DistanceToolCls
//                             NewToolCls
//                             NewTToolCls
//                             SearchToolCls
//                             PickToolCls
//                             InfoWndCls
//                             ToolBarWndCls
//                             CanvasWndCls
//*******************************************************************


#ifndef    __INFO_H
#   define __INFO_H

//note that the following code is executed only
//if this file has not been previously included


#include "chart.h"


_CLASSDEF(SelectionToolCls)
//===================================================================
// class                       SelectionToolCls
//
// Purpose  : Base class for selection button classes.
//
// Notes    :
```

```
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class SelectionToolCls
{
protected:

   HCURSOR hCursor;

public:

   SelectionToolCls()                   {}
   long FindDistance(long, long, long, long);
   int  FindAzimuth(long, long, long, long);

   //abstract methods, which will be implemented in derived classes.
   virtual void DoMouseDown(long, long)  {}
   virtual void DoMouseMove(long, long)  {}
   virtual HCURSOR GetCursor()          {return hCursor;}
   virtual void Reset()                 {}
};


_CLASSDEF(DistanceToolCls)
//=====================================================================
// class                    DistanceToolCls
//
// Purpose   : Behaves for the distance button.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class DistanceToolCls : public SelectionToolCls
{
private:

   BOOL    ClickedBefore;
   long    X1, Y1;
   long    Distance;
   long    Azimuth;
   HCURSOR hCursor1, hCursor2;

public:

   DistanceToolCls(HCURSOR hC1, HCURSOR hC2);
   virtual void DoMouseDown(long X, long Y);
   virtual void DoMouseMove(long X, long Y);
   virtual HCURSOR GetCursor();
   virtual void Reset();
};


_CLASSDEF(NewToolCls)
//=====================================================================
// class                    NewToolCls
//
```

```
// Purpose   : Behaves for the new tool button.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//====================================================================
class NewToolCls : public SelectionToolCls
{
private:

   HCURSOR hCursor1;

public:
   NewToolCls(HCURSOR hC1);
   virtual void DoMouseDown(long X, long Y);
   virtual void DoMouseMove(long X, long Y);
   virtual HCURSOR GetCursor()         {return hCursor1;}
   virtual void Reset();
};


_CLASSDEF(NewTToolCls)
//====================================================================
// class                      NewTToolCls
//
// Purpose   : Behaves for the new from tool button.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//====================================================================
class NewTToolCls : public SelectionToolCls
{
private:

   BOOL    ClickedBefore;
   long    X1, Y1;
   HCURSOR hCursor1, hCursor2;
   long    Distance;
   int     Azimuth;

public:

   NewTToolCls(HCURSOR hC1, HCURSOR hC2);
   virtual void DoMouseDown(long X, long Y);
   virtual void DoMouseMove(long X, long Y);
   virtual HCURSOR GetCursor();
   virtual void Reset();
};


_CLASSDEF(SearchToolCls)
//====================================================================
// class                      SearchToolCls
//
// Purpose   : Behaves for the search button.
//
```

```
// Notes      :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class SearchToolCls : public SelectionToolCls
{
private:

   HCURSOR hCursor1;

public:

   SearchToolCls(HCURSOR hC1);
   virtual void DoMouseDown(long X, long Y);
   virtual void DoMouseMove(long X, long Y);
   virtual HCURSOR GetCursor()         {return hCursor1;}
   virtual void Reset();
};


_CLASSDEF(PickToolCls)
//=================================================================
// class                    PickToolCls
//
// Purpose   : Behaves for the pick tool button.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class PickToolCls : public SelectionToolCls
{
private:

   HCURSOR hCursor1;

public:

   PickToolCls(HCURSOR hC1);
   virtual void DoMouseDown(long X, long Y);
   virtual void DoMouseMove(long X, long Y);
   virtual HCURSOR GetCursor()         {return hCursor1;}
   virtual void Reset();
};


_CLASSDEF(InfoWndCls)
//=================================================================
// class                    InfoWndCls
//
// Purpose   : Displays the info area at the bottom of the window.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
```

```
//=================================================================
class InfoWndCls : public TWindow
{
private:

   RECT        WndRect;
   PTScrollBar HorizantalScroll;

public:

   InfoWndCls(PTWindowsObject AParent, LPSTR ATitle);
   ~InfoWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void WMSize(TMessage&);
   virtual void WMLButtonDown(TMessage &Msg) = [WM_FIRST +
WM_LBUTTONDOWN];
   virtual void MenuONE(RTMessage)   = [CM_FIRST + CM_SCALEONE];
   virtual void MenuTWO(RTMessage)   = [CM_FIRST + CM_SCALETWO];
   virtual void MenuTHREE(RTMessage) = [CM_FIRST + CM_SCALETHREE];
   virtual void MenuFOUR(RTMessage)  = [CM_FIRST + CM_SCALEFOUR];
   virtual void MenuFIVE(RTMessage)  = [CM_FIRST + CM_SCALEFIVE];
   virtual void MenuSIX(RTMessage)   = [CM_FIRST + CM_SCALESIX];
   virtual void WMHScroll(RTMessage Msg);
   virtual void DisplayInfo(char *);
   virtual void DrawScaleButton();
};


_CLASSDEF(ToolBarWndCls)
//=================================================================
// class                        ToolBarWndCls
//
// Purpose   : Displays the tool bar area at the right of the window.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class ToolBarWndCls : public TWindow
{
private:

   RECT              WndRect;
   PTScrollBar       VerticalScroll;
   HBITMAP           hBitmap[5];
   HCURSOR           hCursorFrom, hCursorTo, hCursorPick;
   PSelectionToolCls SelectionArray[5];

public:

   ToolBarWndCls(PTWindowsObject AParent, LPSTR ATitle);
   ~ToolBarWndCls();
   virtual void SetupWindow();
   virtual void Paint(HDC PaintDC, PAINTSTRUCT&);
   virtual void WMSize(TMessage&);
   virtual void WMVScroll(RTMessage Msg);
   virtual void WMLButtonDown(TMessage &Msg) = [WM_FIRST +
WM_LBUTTONDOWN];
};


_CLASSDEF(CanvasWndCls)
//===================================================================
// class                     CanvasWndCls
//
// Purpose   : Displays the drawing area which holds targets, gun
//             positions etc.
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//             All Rights Reserved
//===================================================================
class CanvasWndCls : public TWindow
{
private:

   RECT    WndRect;
   HBITMAP TickMarkBmp;
   HBITMAP TickMarkMask;

public:

   CanvasWndCls(PTWindowsObject AParent, LPSTR ATitle);
   ~CanvasWndCls();
   virtual LPSTR GetClassName();
   virtual void SetupWindow();
   virtual void Paint( HDC PaintDC, PAINTSTRUCT&);
   virtual void WMLButtonDown(TMessage &Msg) = [WM_FIRST +
WM_LBUTTONDOWN];
   virtual void WMMouseMove(TMessage &Msg)   = [WM_FIRST + WM_MOUSEMOVE];
   virtual void WMSetCursor(TMessage &Msg)   = [WM_FIRST + WM_SETCURSOR];
   virtual BOOL ReadTargetData();
   virtual BOOL ReadGunData();
   virtual void DisplayGuns(HDC hDC);
   virtual void DisplayGunLabels(HDC);
   virtual void DrawGrids(HDC hDC, RECT Rect);
   virtual void DisplayTargets(HDC hDC);
   virtual void DisplayTargetLabels(HDC hDC);
   virtual void DrawLocationLines(long, long);
   virtual void TestConstraints();
};

#endif __INFO_H
```

## Listing B.8: TARGETDB.H header file

```
//*******************************************************************
//
//                               targetdb.h
//
// header file for the class TargetDBCls
//*******************************************************************
```

```
#ifndef    _TARGETDB_H
#   define _TARGETDB_H

//note that the following code is executed only
//if this file has not been previously included


#include <string.h>


#defineMAXTARGETNOLEN10
#define MAXCOORLEN      10
#defineMAXDESLEN       50
#defineMAXCATLEN       50
#define MAXBUF          50


_CLASSDEF(TargetDBCls)
//======================================================================
// class                        TargetDBCls
//
// Purpose  : Serves as database management class for target database.
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//           All Rights Reserved
//======================================================================
class TargetDBCls
{
private:

   char  TargetNo[MAXTARGETNOLEN];
   long  CoorEast;
   long  CoorNorth;
   int   Height;
   char  Description[MAXDESLEN];
   char  Category[MAXCATLEN];

public:

   //...................................................................
   TargetDBCls()
   //...................................................................
   {
      TargetNo[0]      = NULL;
      CoorEast         = 0;
      CoorNorth        = 0;
      Height           = 0;
      Description[0]   = NULL;
      Category[0]      = NULL;
   }


   //...................................................................
   TargetDBCls(char* tn,
            long  ea,
            long  no,
            int   he,
            char* des,
            char* cat)
//...............................................................
{
   strcpy(TargetNo, tn);
   CoorEast     = ea;
   CoorNorth    = no;
   Height       = he;
   strcpy(Description, des);
   strcpy(Category, cat);
}


//...............................................................
TargetDBCls(const TargetDBCls& T) //copy initializer
//...............................................................
{
   strcpy(TargetNo, T.TargetNo);
   CoorEast    = T.CoorEast;
   CoorNorth   = T.CoorNorth;
   Height      = T.Height;
   strcpy(Description, T.Description);
   strcpy(Category, T.Category);
}


//...............................................................
~TargetDBCls()
//...............................................................
{}



//...............................................................
TargetDBCls& operator = (const TargetDBCls &T)
//...............................................................
//assignment operator
{
   strcpy(TargetNo, T.TargetNo);
   CoorEast       = T.CoorEast;
   CoorNorth      = T.CoorNorth;
   Height         = T.Height;
   strcpy(Description, T.Description);
   strcpy(Category, T.Category);
   return *this;
 }



//...............................................................
int operator <  (TargetDBCls& aTarget)
//...............................................................
{
   return strcmp(GetTargetNo(), aTarget.GetTargetNo()) < 0;
}
```

```
//.................................................................
int operator >  (TargetDBCls& aTarget)
//.................................................................
{
   return strcmp(GetTargetNo(), aTarget.GetTargetNo()) > 0;
}


//.................................................................
int operator <=  (TargetDBCls& aTarget)
//.................................................................
{
   return strcmp(GetTargetNo(), aTarget.GetTargetNo()) <= 0;
}


//.................................................................
int operator == (TargetDBCls& aTarget)
//.................................................................
{
   return strcmp(GetTargetNo(), aTarget.GetTargetNo()) == 0;
}


//---functions to access the data members
char* GetTargetNo()  {return TargetNo;}
long  GetCoorEast()      {return CoorEast;}
long  GetCoorNorth()   {return CoorNorth;}
int   GetHeight()   {return Height;}
char* GetDescription()  {return Description;}
char* GetCategory()   {return Category;}
//---functions to set the data members
void SetTargetNo(char* tn)     {strcpy(TargetNo, tn);}
void SetCoorEast(long east)  {CoorEast = east;}
void SetCoorNorth(long north)  {CoorNorth = north;}
void SetHeight(int height)   {Height = height;}
void SetDescription(char* des) {strcpy(Description, des);}
void SetCategory(char* cat)  {strcpy(Category, cat);}


//.................................................................
void Reset()
//.................................................................
{
   TargetNo[0]     = NULL;
   CoorEast       = 0;
   CoorNorth      = 0;
   Height         = 0;
   Description[0] = NULL;
   Category[0]    = NULL;
}

};

#endif _TARGETDB_H
```

# APPENDIX C. (DATABASE HEADER FILES LISTINGS)

## Listing C.1: DBGUN.H header file

```
//*****************************************************************
//
//                              dbgun.h
//
// header file for the class DBGunCls
//*****************************************************************



#ifndef    _DBGUN_H
#   define _DBGUN_H



#include "WStr.h"
#include "Paradox.h"



#define MAXNAMELEN   10
#define MAXMODELLEN  30



_CLASSDEF(DBGunCls)
//=================================================================
// class                       DBGunCls
//
// Purpose   : Performs database management for guns.
// Notes     :
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=================================================================
class DBGunCls
{
private:
   //#1
   char           BattalionName[MAXNAMELEN];
   char           BatteryName[MAXNAMELEN];
   short          GunNo;
   long           CoorEast;
   long           CoorNorth;
   int            Height;
   BOOL           MainGun;
   char           Model[MAXMODELLEN];
   long           MaxRange;
   int            MaxRightDeflection;
   int            MaxLeftDeflection;
   int            CommonOrientationDeflection;
   int            OrientationDeflection;
   char           FiringTableName[MAXNAMELEN];
   //#14

   //Paradox table info
   ParadoxTable TableGuns;
   int            nFields;
   LPSTR          Names[14];
   LPSTR          Types[14];

protected:

   void RecordToBuffer();

public:

   DBGunCls(LPSTR TableName);
   ~DBGunCls();

   BOOL ReadFirst();
   BOOL ReadLast();
   BOOL ReadNext();
   BOOL ReadPrev();
   BOOL isTableEmpty();
   BOOL isLast();

   BOOL AddRecord();
   BOOL UpdateRecord();
   BOOL DeleteRecord();
   BOOL SearchByKey(char *Btn, char *Btr, int Gun);

   void Reset();

   //functions to access the data members
   char *GetBattalionName()        {return BattalionName;}
   void  SetBattalionName(char *in){strcpy(BattalionName, in);}

   char *GetBatteryName()          {return BatteryName;}
   void  SetBatteryName(char *in)  {strcpy(BatteryName, in);}

   short GetGunNo()                {return GunNo;}
   void  SetGunNo(short in)        {GunNo = in;}

   long  GetCoorEast()             {return CoorEast;}
   void  SetCoorEast(long in)      {CoorEast = in;}

   long  GetCoorNorth()            {return CoorNorth;}
   void  SetCoorNorth(long in)     {CoorNorth = in;}

   int   GetHeight()               {return Height;}
   void  SetHeight(int in)         {Height = in;}

   BOOL  isMainGun()               {return MainGun;}
   void  SetMainGun(BOOL in)       {MainGun = in;}

   char *GetModel()                {return Model;}
   void  SetModel(char *in)        {strcpy(Model, in);}

   long  GetMaxRange()             {return MaxRange;}
   void  SetMaxRange(long in)      {MaxRange = in;}

   int   GetMaxRightDeflection()   {return MaxRightDeflection;}
   void  SetMaxLeftDeflection(int in){MaxLeftDeflection = in;}

   int   GetMaxLeftDeflection()    {return MaxLeftDeflection;}
   void  SetMaxRightDeflection(int in){MaxRightDeflection = in;}
   int   GetCommonOrientation()
            {return CommonOrientationDeflection;}
   void  SetCommonOrientation(int in)
            {CommonOrientationDeflection = in;}

   int   GetOrientationDeflection()        {return OrientationDeflection;}
   void  SetOrientationDeflection(int in)  {OrientationDeflection = in;}

   char *GetFiringTableName()              {return FiringTableName;}
   void  SetFiringTableName(char *fn)      {strcpy(FiringTableName, fn);}

};

#endif _DBGUN_H
```

## Listing C.2: DBTARGET.H header file

```
//*********************************************************************
//
//                              dbtarget.h
//
// header file for the class DBTargetCls
//*********************************************************************



#ifndef    _DBTARGET_H
#   define _DBTARGET_H



#include "WStr.h"
#include "Paradox.h"


#define SUCCESS          1
#define FAIL             0
#define MAXTARGETNOLEN  10
#define MAXCOORLEN      10
#define MAXDESLEN       50
#define MAXCATLEN       50
#define MAXBUF          50


_CLASSDEF(DBTargetCls)
//=====================================================================
// class                        DBTargetCls
//
// Purpose   : Performs database management for targets.
// Notes     :
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class DBTargetCls
{
private:
   char   TargetNo[MAXTARGETNOLEN];
   double CoorEast;
   double CoorNorth;
   double Height;
   char   Description[MAXDESLEN];
   char   Category[MAXCATLEN];

   //Paradox table info
   ParadoxTable TableTargets;
   int             nFields;
   LPSTR           Names[6];
   LPSTR           Types[6];

protected:

   void RecordToBuffer();
   void BufferToRecord();

public:

   DBTargetCls(LPSTR TableName);
   ~DBTargetCls();

   BOOL ReadFirst();
   BOOL ReadLast();
   BOOL ReadNext();
   BOOL ReadPrev();
   BOOL isTableEmpty();
   BOOL isLast();

   BOOL AddRecord();
   BOOL UpdateRecord();
   BOOL DeleteRecord();
   BOOL SearchByKey(char *Key);

   void Reset();

   char   *GetTargetNo()            {return TargetNo;}
   void    SetTargetNo(char* tn)    {strcpy(TargetNo, tn);}

   double GetCoorEast()             {return CoorEast;}
   void   SetCoorEast(double east){CoorEast = east;}

   double GetCoorNorth()              {return CoorNorth;}
   void   SetCoorNorth(double north){CoorNorth = north;}

   double GetHeight()               {return Height;}
   void   SetHeight(double height){Height = height;}

   char   *GetDescription()           {return Description;}
   void    SetDescription(char* des)  {strcpy(Description, des);}

   char   *GetCategory()            {return Category;}
   void    SetCategory(char* cat)   {strcpy(Category, cat);}
};

#endif _DBTARGET_H
```

## Listing C.3: DBTBL_F.H header file

```
//*********************************************************************
//
//                              dbtbl_f.h
//
// header file for the clas DBTable_F_Cls
//*********************************************************************


#ifndef    __DBTABLE_F_H
#   define __DBTABLE_F_H


#include "WStr.h"
#include "Paradox.h"


_CLASSDEF(DBTable_F_Cls)
//=====================================================================
// class                        DBTable_F_Cls
//
// Purpose   : Performs database management for Table F of firing table
// Notes     :
// Copyright: Copyright (c) 1993, Mustafa ESER
//              All Rights Reserved
//=====================================================================
class DBTable_F_Cls
{
private:

   double Range;                //0
   double Elevation;            //1
   double FSforGrazeBurst;      //2
   double DFSper10mDecHOB;      //3
   double DRper1milDElev;       //4
   double Fork;                 //5
   double TimeOfFlight;         //6
   double AzimuthCorrDrift;     //7
   double AzimuthCorrCrossWind;//8
   double MuzzleVelocityDec;    //9
   double MuzzleVelocityInc;    //10
   double RangeWindHead;        //11
   double RangeWindTail;        //12
   double AirTempDec;           //13
   double AirTempInc;           //14
   double AirDensityDec;        //15
   double AirDensityInc;        //16
   double ProjectileWeightDec; //17
   double ProjectileWeightInc; //18

   //Paradox table info
   ParadoxTable Table_F;
   int             nFields;
   LPSTR           Names[18];
   LPSTR           Types[19];
protected:

   void RecordToBuffer();

public:

   DBTable_F_Cls(LPSTR TableName);
   ~DBTable_F_Cls();

   BOOL ReadFirst();
   BOOL ReadLast();
   BOOL ReadNext();
   BOOL ReadPrev();
   BOOL isTableEmpty();
   BOOL isLast();

   void Reset();

   double FindElevation(double range);
};


#endif __DBTABLE_F_H
```

## Listing C.4: PARADOX.H header file

```
//**********************************************************************
//
//                              paradox.h
//
// header file for database engine inclusions
//**********************************************************************


#ifndef PARADOX_H
#   define PARADOX_H


#   include <alloc.h>
#   include <string.h>
#   include "PdoxEnging.h"
#   include "PdoxTable.h"
#   include "PdoxRecord.h"

// Note the following code is only executed if
// this file has not been previously included


char *GetParadoxError (int nError);


#endif
```

## Listing C.5: PDOXENG.H header file

```
//*******************************************************************
//
//                              pdoxeng.h
//
// header file for the class SelectionWndCls
//*******************************************************************



#ifndef    ParadoxEngine_H
#   define ParadoxEngine_H

// Note the following code is only executed if
// this file has not been previously included



#include "pxengine.h"

#ifdef __BORLANDC__
   //PC computer specific includes
#   include <owl.h>
#endif



#ifdef Unix
   // Unix computer specific includes
#endif



//===================================================================
// class                     ParadoxEngine
//
// Purpose  :
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//===================================================================
_CLASSDEF (ParadoxEngine)
class ParadoxEngine
{
public:

   static Status;

   ParadoxEngine (LPSTR szAPlication);
   ~ParadoxEngine ();
};


#endif ParadoxEngine_H
```

## Listing C.6: PDOXREC.H header file

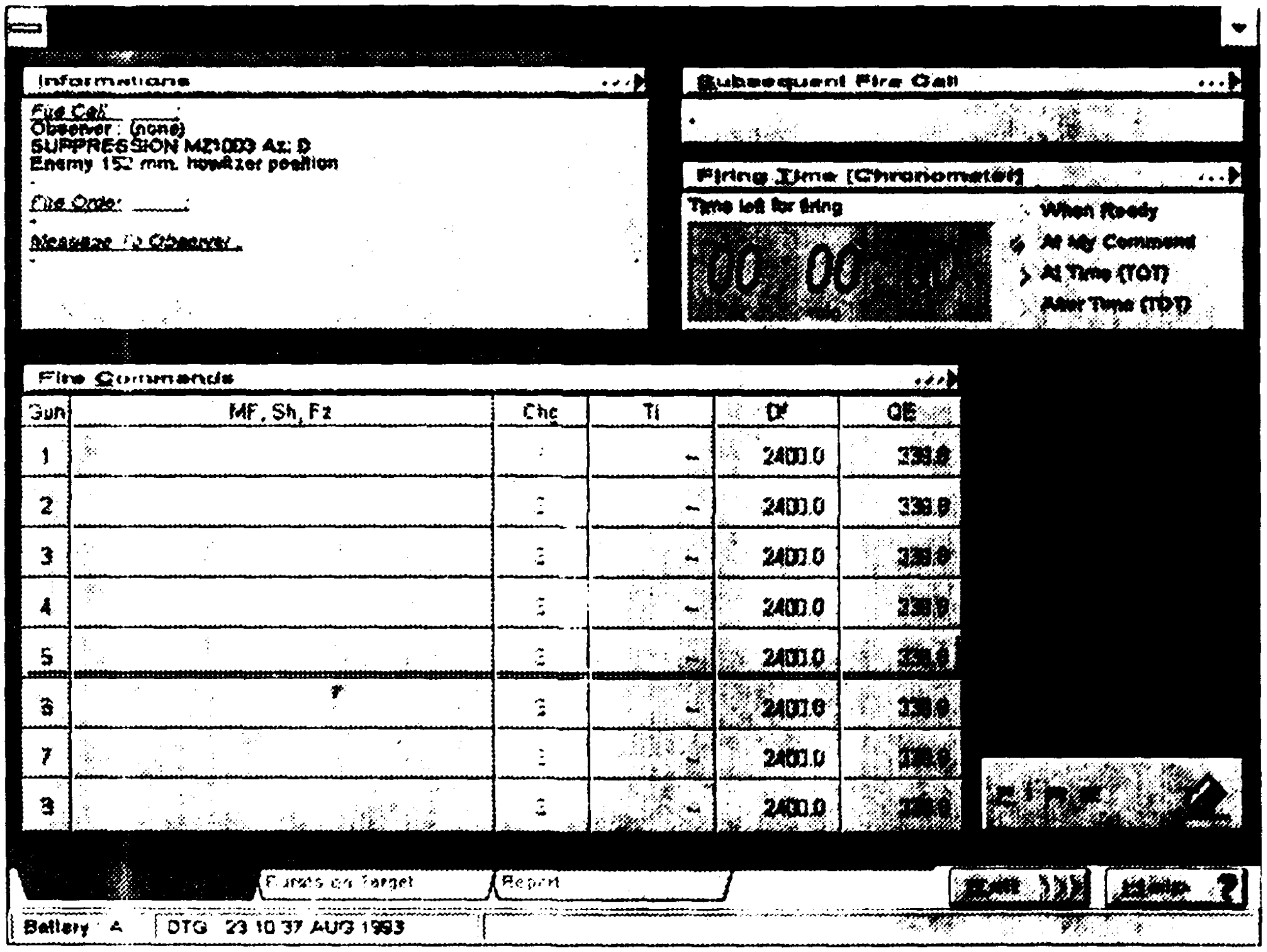
```
//*******************************************************************
//
//                              pdoxrec.h
//
// header file for the class ParadoxRecord
//*******************************************************************



#ifndef    ParadoxRecord_H
#   define ParadoxRecord_H

// Note the following code is only executed if
// this file has not been previously included



#include <ctype.h>
#include "PxEngine.h"
#include "WStr.h"



#ifdef __BORLANDC__
// PC computer specific includes
#include <owl.h>
#endif



#ifdef Unix
// Unix computer specific includes
#endif



_CLASSDEF (ParadoxTable)
_CLASSDEF (ParadoxRecord)
//==================================================================
// class                        ParadoxRecord
//
// Purpose   :
//
// Notes     :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//==================================================================
class ParadoxRecord
{
private:

   ParadoxTable ParentTable;
public:

   RECORDHANDLE HRecord;
   intStatus;

   ParadoxRecord (ParadoxTable);
   virtual ~ParadoxRecord ();

   void RawGet (void *, int BufferSize);
   void RaPut (void *, int BufferSize);

   WStr GetField (FIELDHANDLE FieldNumber);
   WStr GetField (LPSTR FieldName);
   void PutField (FIELDHANDLE FieldNumber, WStr);
   void PutField (LPSTR FieldName, WStr);

   int Update ();
   int Add ();
   int Read ();
};


#include "PdoxTable.h"


#endif ParadoxRecord_H
```

## Listing C.7: PDOXTAB.H header file

```
//***********************************************************************
//
//                               Pdoxtab.h
//
// header file for the class ParadoxTable
//***********************************************************************


#ifndef    ParadoxTable_H
#   define ParadoxTable_H

// Note the following code is only executed if
// this file has not been previously included


#include <DIR.H>
#include "WStr.h"
#include "pxengine.h"
#include "PdoxEngine.h"
#include "PdoxRecord.h"


#ifdef __BORLANDC__
```

```
// PC computer specific includes
#include<owl.h>
#endif



#ifdef Unix
// Unix computer specific includes
#endif



_CLASSDEF (ParadoxTable)
//=====================================================================
// class                         ParadoxTable
//
// Purpose  :
//
// Notes    :
//
// Copyright: Copyright (c) 1993, Mustafa ESER
//            All Rights Reserved
//=====================================================================
class ParadoxTable
{
public:

   intStatus;
   ParadoxRecord Record;
   TABLEHANDLE     HTable;

   ParadoxTable (LPSTR  TableName,
                 int    IndexField = 0,
                 int    nFields = 0,
                 char   *Names[] = NULL,
                 char   *Types[] = NULL);
   virtual ~ParadoxTable ();

   void DeleteAll ();

   RECORDNUMBER TotalRecords ();
   RECORDNUMBER CurrentRecord ();

   void RecordUpdate ();
   void RecordDelete ();
   void RecordAdd ();

   void ReadFirst ();
   void ReadLast ();
   void ReadNext ();
   void ReadPrev ();
   void ReadRecord (RECORDNUMBER RecordNumber);

   void IndexDelete (char *Field);
   void IndexDelete (FIELDHANDLE FieldNumber);
   void IndexAdd (char *Field, int Mode = INCSECONDARY);
   void IndexAdd (FIELDHANDLE FieldNumber, int Mode = INCSECONDARY);

   void Find (WStr Match, LPSTR FieldName, int Mode = SEARCHFIRST);
  void Find (WStr Match, FIELDHANDLE FieldNumber, int Mode = SEARCHFIRST);
);
```

[illegible]



Fire Call:
Observer : (none)
SUPPRESSION MZ1003 Az: 0
Enemy 152 mm. howitzer position

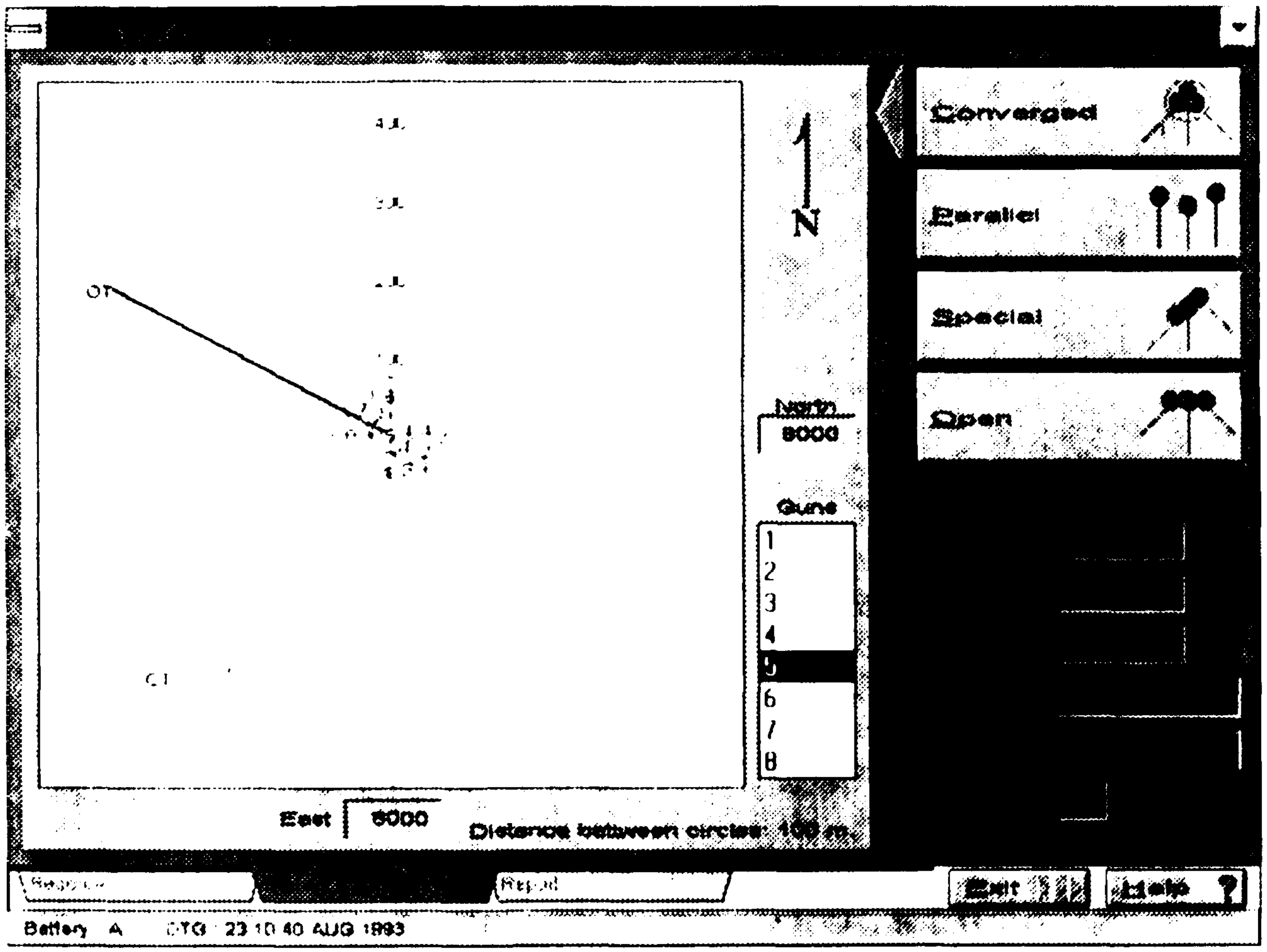
Fire Order: ____ :

Message To Observer :

| Gun | MF, Sh, Fz | Chg | Ti | Df | QE |
|---|---|---|---|---|---|
| 1 | | 3 | - | 2400.0 | 338.0 |
| 2 | | 3 | - | 2400.0 | 338.0 |
| 3 | | 3 | - | 2400.0 | 338.0 |
| 4 | | 3 | - | 2400.0 | 338.0 |
| 5 | | 3 | - | 2400.0 | 338.0 |
| 6 | | 3 | - | 2400.0 | 338.0 |
| 7 | | 3 | - | 2400.0 | [illegible] |
| 8 | | 3 | - | 2400.0 | [illegible] |

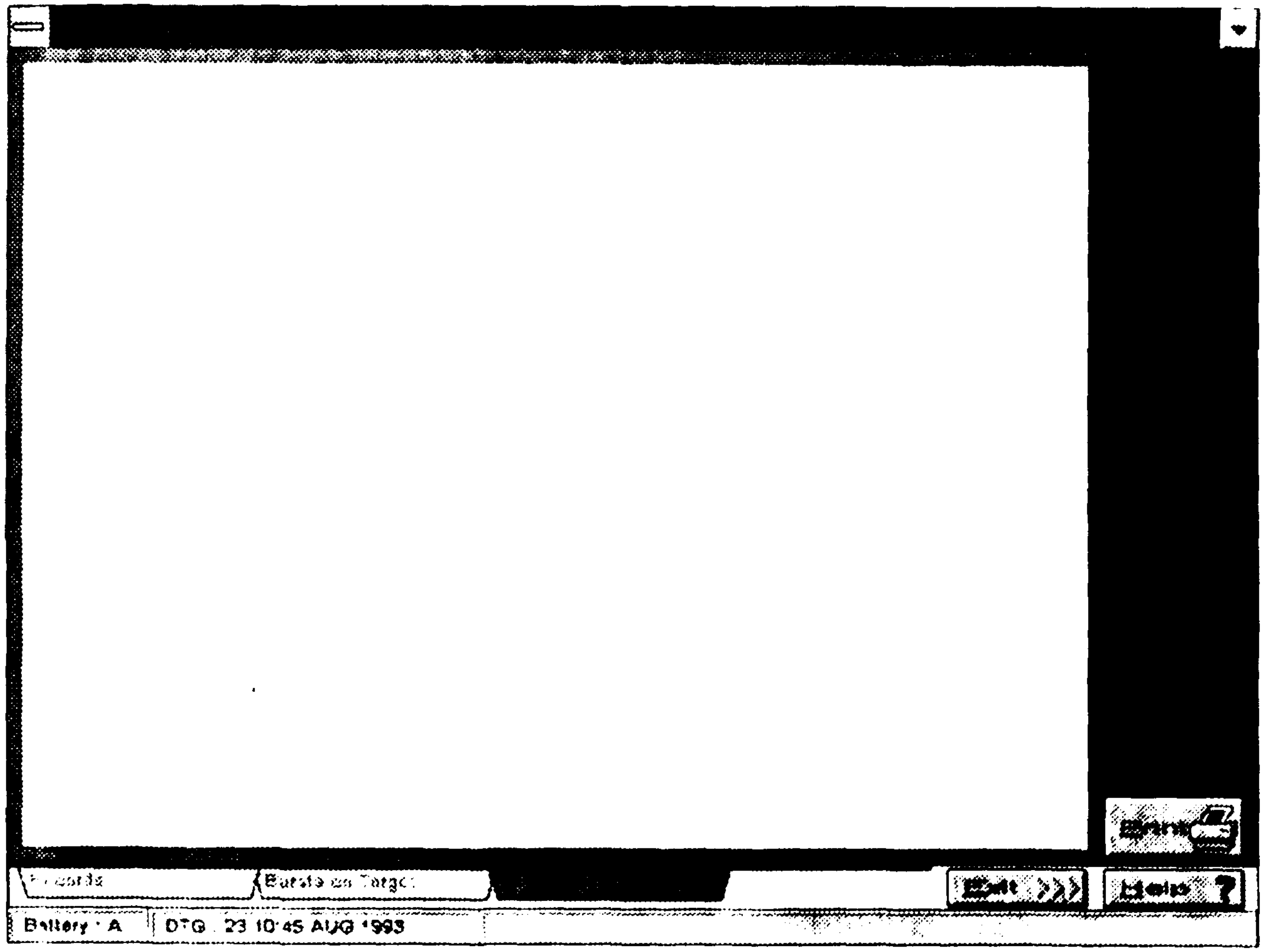
Battery : A | DTG 23 10 37 AUG 1993

Screen E: (Other) First page of the Record of Fire screen.

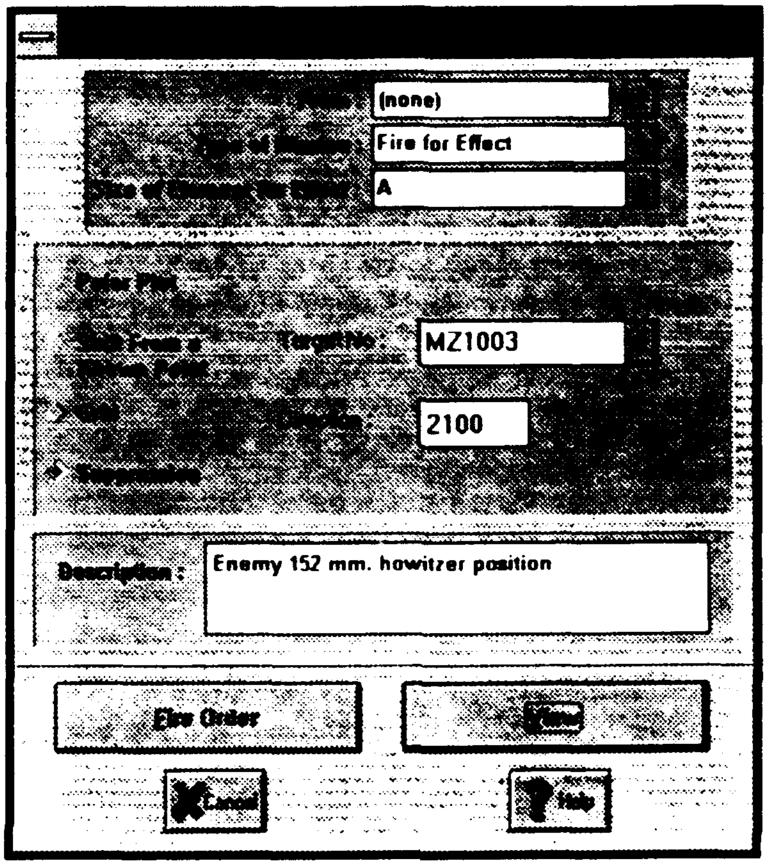


Battery A DTG : 23 10 40 AUG 1993

Screen 2: (OFire)Second page of the Record of Fire screen



Screen 3: (OFire) Third page of the Record of Fire Screen.



(none)

Fire for Effect

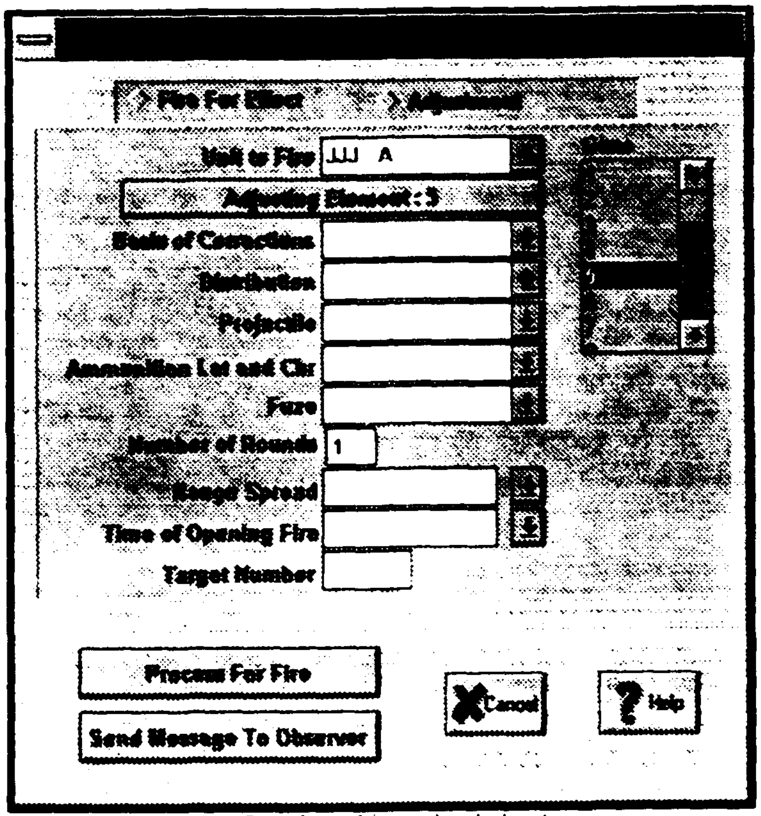
A

Target No.: MZ1003

2100

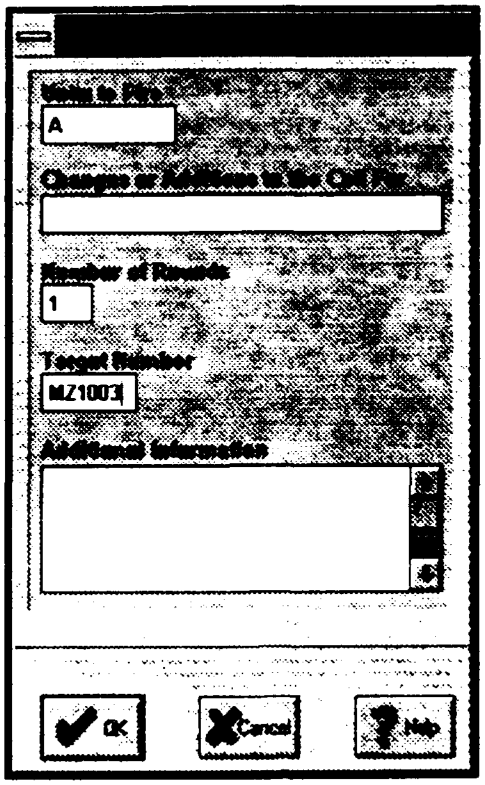
Description : Enemy 152 mm. howitzer position

Screen 4: (OFire) Call for fire dialog box.



Fire For Effect
Adjustment
Unit to Fire ⊥⊥⊥ A
Adjusting Element : 3
Basis of Corrections
Distribution
Projectile
Ammunition Lot and Chr
Fuze
Number of Rounds 1
Range/ Spread
Time of Opening Fire
Target Number
Process For Fire
Send Message To Observer
Cancel
Help

Screen 5: (OFire) Fire order dialog box.



Units to Fire
A

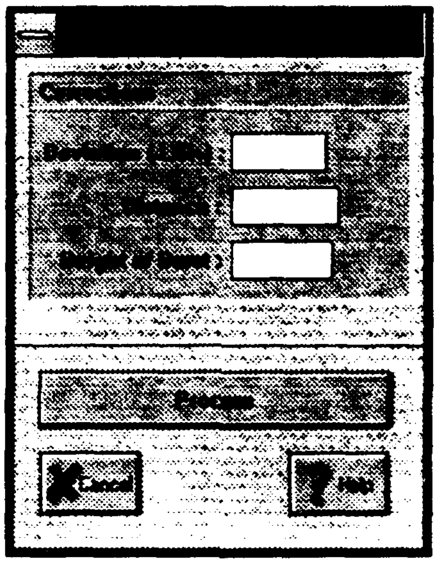
Changes or Additions to the Call For

Number of Rounds
1

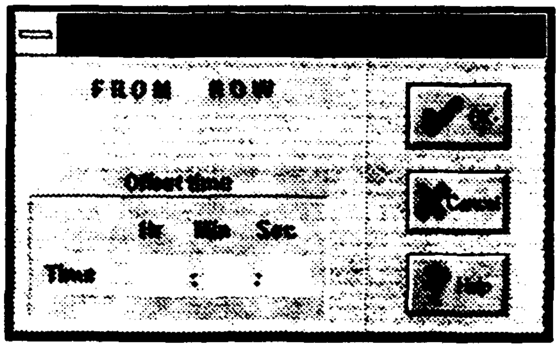
Target Number
MZ1003

Additional Information

Screen 6: (OFire) Message to observer dialog box.

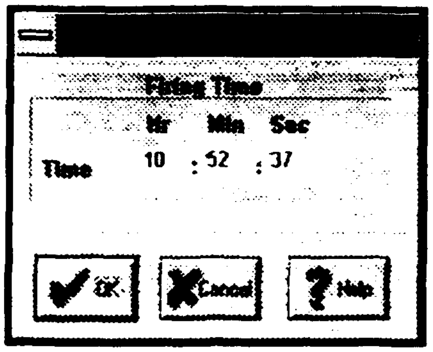


**Screen 7:** (OFire) Subsequent fire call dialog box.



Offset time

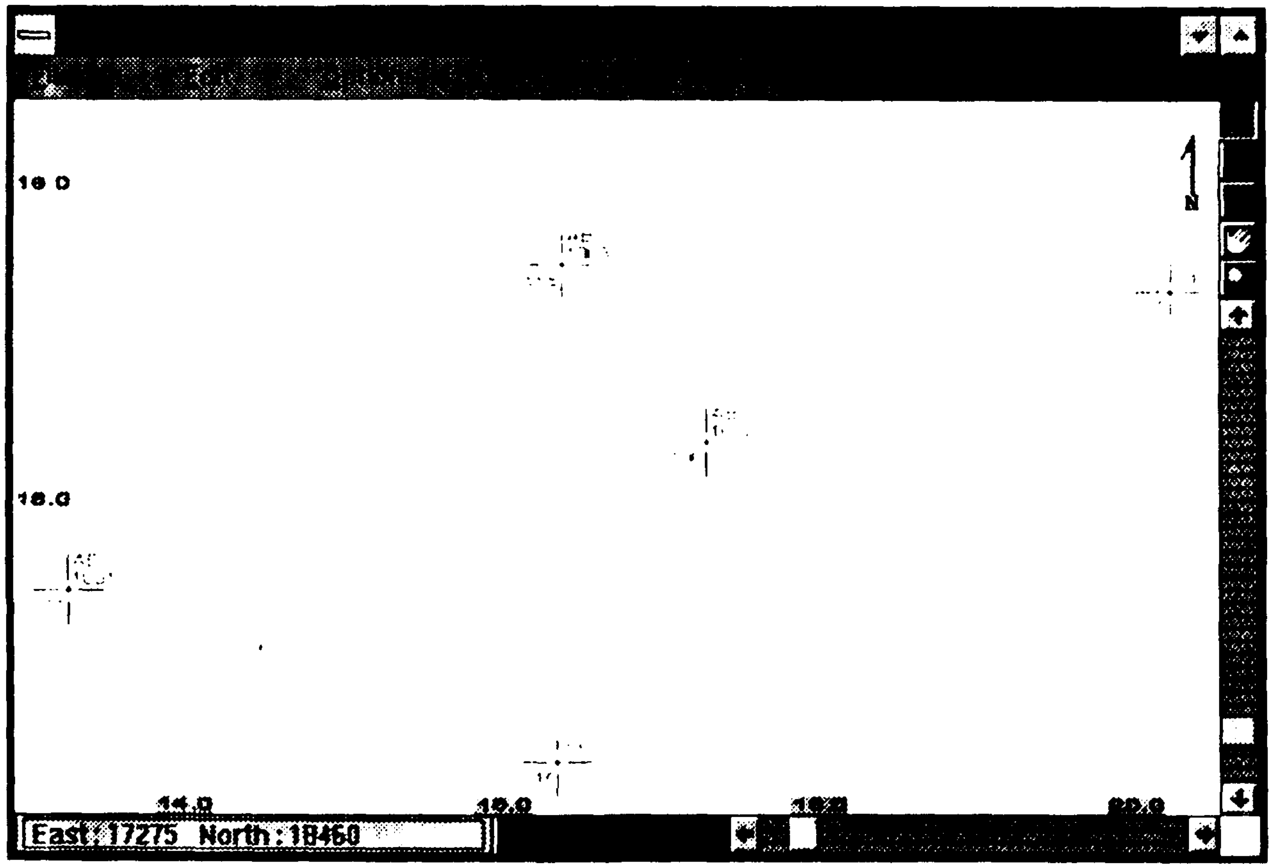
| | Hr | Min | Sec |
|---|---|---|---|
| Time | : | : | |



Firing Time

| | Hr | Min | Sec |
|---|---|---|---|
| Time | 10 : | 52 : | 37 |

**Screen 8:** (OFire) Time-on-target dialog boxes. The above dialog box set the time refering the present time, while the below one sets the exact time.



East : 17275   North : 18460

Screen 9: (OChart) The firing chart window.

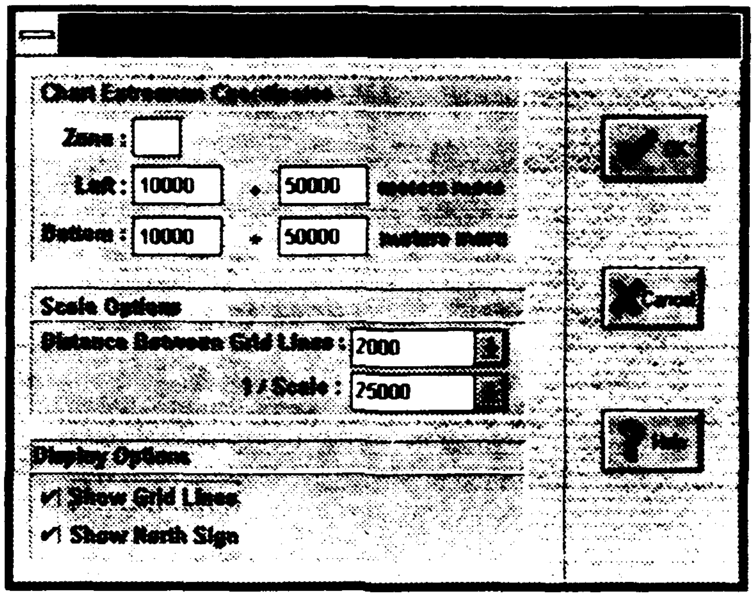


Chart Extremum Coordinates

Zone :

Left : 10000 ⇔ 50000 [illegible]

Bottom : 10000 ⇔ 50000 [illegible]

Scale Options

Distance Between Grid Lines : 2000

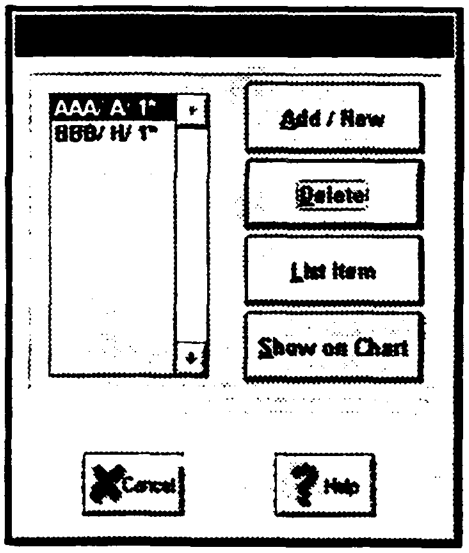
1 / Scale : 25000

Display Options

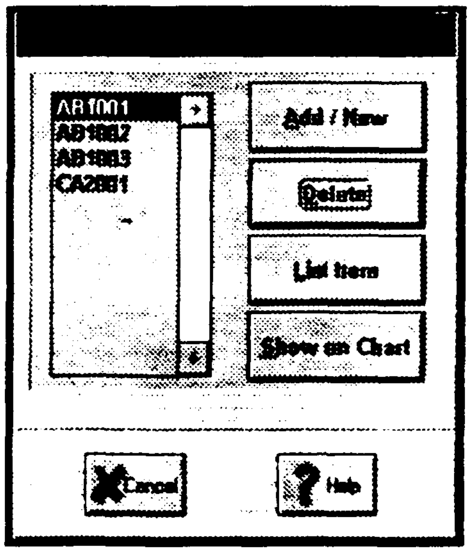
✓ Show Grid Lines

✓ Show North Sign

Screen 10: (OChart) Grid setup dialog box.

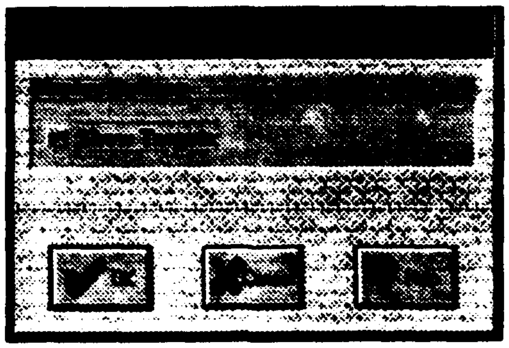


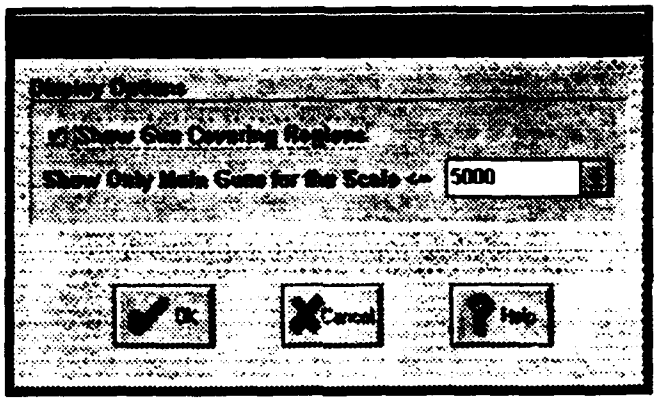
AAA/ A/ 1*
BBB/ H/ 1*



AB1001
AB1002
AB1003
CA2001

Screen 11: (OChart) Both target and guns database dialog boxes.

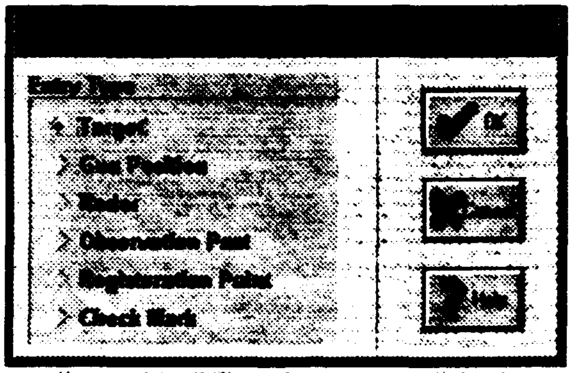

Screen 12: (OChart) Target display setup dialog box.


Display Options

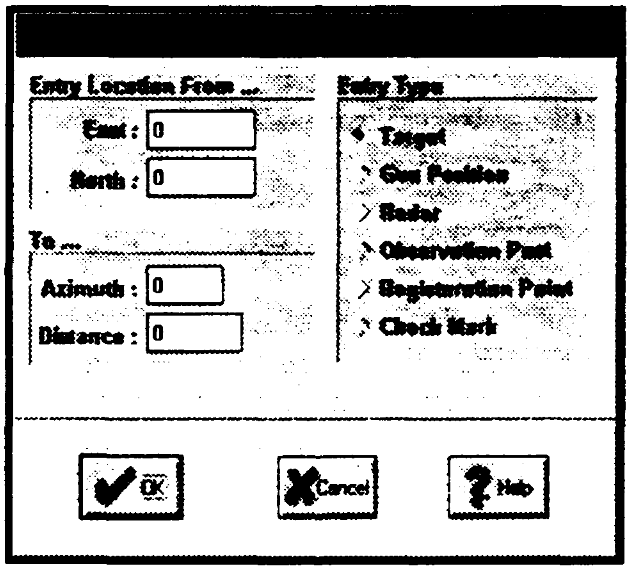
[x] Show Gun Covering Regions

Show Only Main Guns for the Scale <= 5000

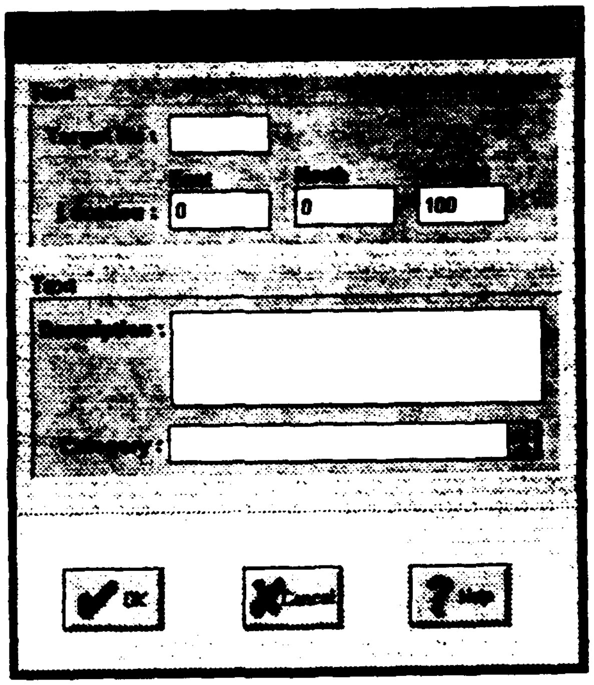
Screen 13: (OChart) Gun display setup dialog box.



Screen 14: (OChart) Input inquiry dialog box.

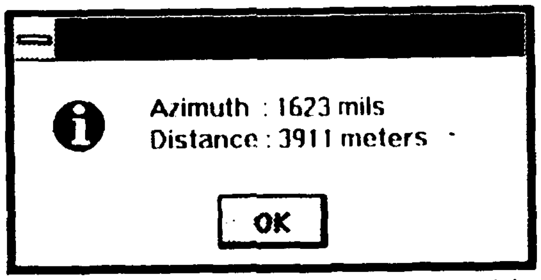


Screen 15: (OChart) Input inquiry dialog box (refering to a point)



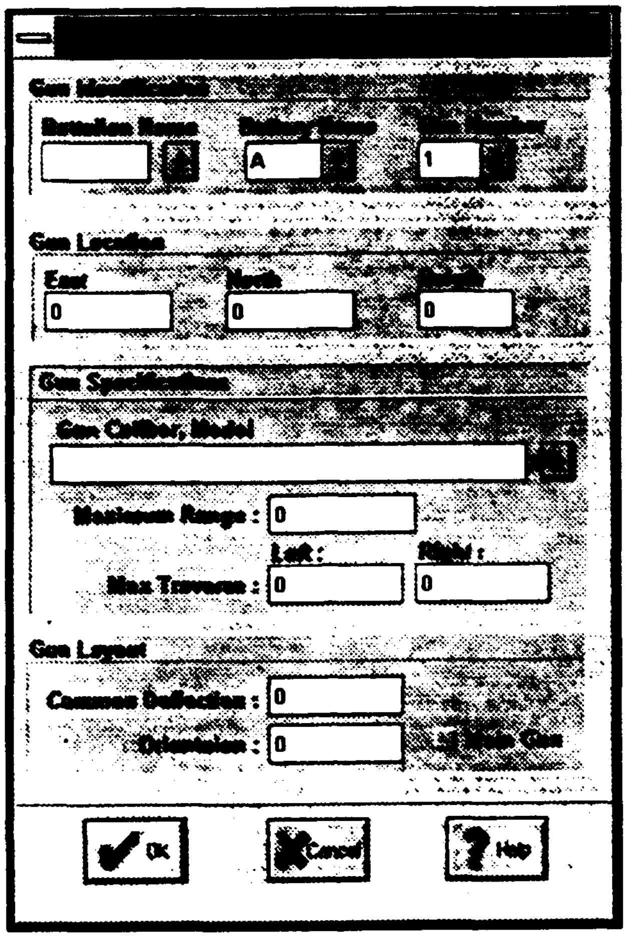
0 0 100

Screen 16: (OChart) Target input dialog box.



Azimuth : 1623 mils
Distance : 3911 meters

Screen 17: Azimuth and distance distance display dialog box.


**Gun Identification**

Battalion Name:

Battery Name: A

Gun Number: 1

**Gun Location**

East: 0

North: 0

Altitude: 0

**Gun Specifications**

Gun Caliber, Model:

Maximum Range : 0

Max Traverse : Left : 0 Right : 0

**Gun Layout**

Common Deflection : 0

Orientation : 0

Base Gun

Screen 18: (OChart) Gun input dialog box.

# LIST OF REFERENCES

[ 1 ] Andleigh, P. K. and Gretzinger, M. R., *Distributed Object-Oriented Data-Systems Design*, Prentice-Hall, NJ, 1992.

[ 2 ] Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, NJ, 1981.

[ 3 ] Booch, G., *Software Engineering with ADA*, The Benjamin/Cumming Publishing Company, CA, 1986.

[ 4 ] Booch, G., *Object-Oriented Design with Applications*, The Benjamin/ Cummings Publishing Company, Inc., CA, 1991.

[ 5 ] Chorafas, D., *The Software Handbook*, Petrocelli Books, NJ, 1984.

[ 6 ] Coad, P., and Yourdon, E., *Object-Oriented Analysis*, 2nd edition, Prentice Hall, N.J., 1991.

[ 7 ] Department of the Army, *FM 6-20-1, Tactics, Techniques, and Procedures for the Field Artillery Cannon Battalion*, 3 November 1989.

[ 8 ] Department of the Army, *FM 6-20-2, Tactics, Techniques, and Procedures for Corps Artillery, Division Artillery, and Field Artillery Brigade Headquarters*, 7 Jan 1993.

[ 9 ] Department of the Army, *FM 6-30, Tactics, Techniques, and Procedures for Observed Fire*, 16 July 1991.

[ 10 ] Department of the Army, *FM 6-50, Tactics, Techniques, and Procedures for the Field Artillery Cannon Battery*, 18 May 1990.

[ 11 ] Department of the Army, *FM 6-40, Field Artillery, Manual Cannon Gunnery*, 8 April 1988.

[ 12 ] Department of the Army, *FM 21-26, Map Reading and Land Navigation*,30 September 1987.

[ 13 ] Elmasri, R. and Shamkant, B. N., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989.

[ 14 ] Fichman, R. G. and Kemerer, C. F., "Object-Oriented and Conventional Analysis and Design Methodologies", *Computer*, pp.22-39, October 1992.

[ 15 ] Jacobson, I., "Is Object Technology Software's Industrial Platform?", *IEEE Software*, pp.24-30, January 1993.

[ 16 ] Pittman, M., "Lessons Learned in Managing Object-Oriented Development, *IEEE Software*, pp.43-53, January 1993.

[ 17 ] Porter, A., *C++ Programming for Windows*, Osborne McGraw-Hill, CA, 1993.

[ 18 ] Rao, R., B., *C++ and the OOP Paradigm*, McGraw-Hill, Inc., NY, 1993.

[ 19 ] Roetzheim, W. H., *Programming Windows with Borland C++*, Ziff-Davis Press, CA, 1992.

[ 20 ] Roetzheim, W. H., *Uncharted Windows Programming*, Sams Publishing, IN, 1993.

[ 21 ] Ross, D.T., Goodenough, J.B., Irvine, C.A., "Software Engineering: Process, Principles and Goals", *Computer*, May 1975.

[ 22 ] Stevens, A., *C++ Database Development*, MIS: Press, NY, 1992.

[ 23 ] Stiles, E. J., "AFATDS- It's not a a New TACFIRE", *Field Artillery*, pp. 39-41, February 1992.

[ 24 ] Stroustup, B., *The C++ Programming Language*, Addison-Wesley Publishing Company, N.J., 1992.

[ 25 ] Syck, G., *Object Windows How-to*, Waite Group Press, CA, 1993.

[ 26 ] Thomson, D., "Interfacing Objects with Relational DBMS", *Database Programming & Design*, v.6, pp.32-41, August 1993.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Dudley Knox Library<br>Code 52<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 2 |
| 3. | Chairman, Code CS<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943 | 2 |
| 4. | Dr. C. Thomas Wu<br>Code CS/Wq<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943 | 2 |
| 5. | Dr. David K. Hsiao<br>Code CS/Hs<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943 | 1 |
| 6. | Kara Kuvvetleri Komutanligi<br>Kutuphanesi<br>Bakanliklar / ANKARA<br>TURKEY | 1 |
| 7. | Kara Harp Okulu<br>Kutuphanesi<br>Dikmen / ANKARA<br>TURKEY | 1 |

8. Mustafa Eser 3
Evsat Mah. Ozdilek cad. 11.Sok. No:6A
42700 Beysehir / KONYA
TURKEY

9. Top. Kd.Utgm. Turgay Cince 1
Sukraniye mah. Yuksel sk. No:34
BURSA
TURKEY